VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

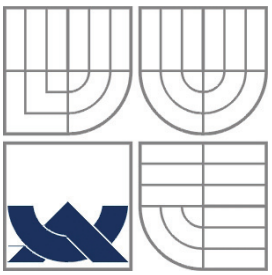
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO PODPORU POSUZOVÁNÍ ZPŮSOBILOSTI PROCESŮ VÝVOJE SOFTWARU

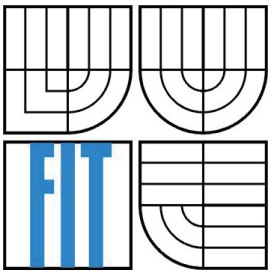
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JOSEF STRAPÁČ
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO PODPORU POSUZOVÁNÍ ZPŮSOBILOSTI PROCESŮ VÝVOJE SOFTWARU

SYSTEM FOR CAPABILITY SUPPORT OF SOFTWARE DEVELOPMENT PROCESSES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JOSEF STRAPÁČ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. KVĚTOŇOVÁ ŠÁRKA

BRNO 2009

Abstrakt

Posuzování způsobilosti procesů při vývoji softwaru úzce souvisí s teorií managementu jakosti. Diplomová práce řeší problematiku řízení procesů během vývoje z několika směrů. Teoretická část se zabývá výkladem pojmů, které jsou spjaty s principy a postupy řízení jakosti. Praktická část je zaměřena na technologické aspekty nutné pro realizaci prototypu systému. Dále obsahuje analýzu a návrh systému, popis implementace a závěrečné zhodnocení.

Abstract

Evaluation of processes during software development is closely associated with theory of quality management. This master‘s thesis deals with problems of quality management during development from various points of view. Theoretical part is involved in definitions of terms which relates to fundamentals and methods of quality management. Practical part is aimed at technological aspects of prototype realization. Onward contains analysis and system design, implementation description and final resume.

Klíčová slova

proces, kvalita, jakost, řízení jakosti, vývoj, návrh, ISO 9000, CMMI, nástroje řízení jakosti, webová aplikace, Flex, Actionscript, MXML, MySQL, Mate framework

Keywords

process, quality, quality management, development, design, ISO 9000, CMMI, quality management tools, web appliacation, Flex, Actionscript, MXML, MySQL, Mate framework

Citace

Strapáč Josef: Systém pro podporu posuzování způsobilosti procesů vývoje softwaru, diplomová práce, Brno, FIT VUT v Brně, 2009

Systém pro podporu posuzování způsobilosti procesů vývoje softwaru

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením inženýrky Šárky Květoňové.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Josef Strapáč
25. května 2009

Poděkování

Na tomto místě bych rád poděkoval Ing. Šárce Květoňové za vedení mé práce a za možnost konzultace, dále rodině za podporu nejen během studia a v neposlední řadě Simonce, mé slečně, za všechno.

Obsah

1 Úvod

Softwarový produkt je jen zdánlivě produkt jako každý jiný. Jelikož SW systém neoplývá fyzikálními vlastnostmi, tak je zřejmé, že i „život“ (možná lépe – životní cyklus) tohoto produktu neodpovídá klasickému průběhu jako u výrobků, které známe ze svého okolí. Nejen tento aspekt, a tím je virtuální existence, je totiž podstatný proto, proč je rozmezí mezi počátkem (ve smyslu počátku užívání) a zánikem softwarových systémů poměrně krátké, než bývá zvykem u produktů inženýrství mimo softwarového. Je to dáno zejména stále měnícími se požadavky na softwarové systémy. Rostou nároky na výkon, bezpečnost a hlavně na kvalitu produktu. Dům, který byl postaven před sto lety může být stejně krásný a funkční, v některých případech i hodnotnější, i po uplynutí takto dlouhé doby. U softwarových systémů je tomu jinak. Již v okamžiku nasazení systém začíná „stárnout“ a je potřeba plánovat kroky, které povedou k vylepšení vlastností a vyhovění požadavkům u dalších verzí systému.

Fáze, které předchází nasazení a uvedení do provozu SW systému, jsou proto pro tento systém nejdůležitější z pohledu kvalitní analýzy a stejně tak z pohledu kvalitního návrhu systému. Tyto kroky je možné chápat jako procesy, které vedou k dosažení výsledku. Z hlediska výsledné kvality SW produktu jsou právě tyto procesy při vývoji softwaru ty podstatné. To, jakým způsobem budou optimalizovány, jakým způsobem budou kooperovat s jinými procesy při vývoji a to, jakým způsobem se je nám podaří zkvalitnit, se ve výsledku projeví jako spokojenost nebo naopak nespokojenost u zadavatele.

Tím, jakým způsobem fungují procesy uvnitř organizace[1], se zabývají různé normy a modely, které jsou úzce spjaty s řízením kvality. Některé udávají standard, jiné jen doporučení. Ve výsledku je na zvážení dané organizace, který způsob hodnocení kvality jim nejvíce vyhovuje, odpovídá jejich potřebám a potřebám zákazníkům. Není žádným tajemstvím, že výsledky, zpravidla vyjádřeny v obratech, jsou měřítkem úspěšného a hlavně kvalitního softwarového produktu.

Kvalitou procesů při vývoji softwaru se zabývá tato diplomová práce. Mimo to je dalším úkolem zhodnocení současných možností nástrojů a prostředků pro vývoj webových aplikací a výběr vhodného prostředí pro realizaci prototypu zahrnující implementaci navržených metrik.

Práce nenavazuje na semestrální projekt a je rozdělena do dvou větších logických celků – teoretického a praktického.

Teoretická část diplomové práce obsahuje několik kapitol tematicky rozdělených podle oblasti, kterou popisují. V úvodní kapitole jsou objasněny základní pojmy jako: management jakosti, proces a kvalita, s tím jsou spojeny kvalitativní charakteristiky softwarových produktů. Normy ISO řady 9000, model CMMI a také Six Sigma, strategie řízení, se nachází v kapitole číslo 3. Tím získáme alespoň základní pohled na tento business sektor. Metody a nástroje řízení jakosti jsou rozebrány v kapitole čtvrté. Jde o kompletní popis dvou skupin nástrojů řízení jakosti tak, jak jsou uváděny v praxi a v různých literaturách ([11], [12]). Kapitola 5 je zaměřena na teorii životního cyklu, tedy fáze, které jsou shodné pro majoritní část softwarových produktů, a modely, kde jsou některé typy důležité jen z hlediska historického, a modely tradiční, moderní, osvědčené a na závěr i modely agilního charakteru.

Úvodem k praktické části je kapitola 6. Zabývá se nástroji a prostředky pro vývoj webových aplikací. Výběr technologií je koncipován tak, aby odpovídal současným trendům, potřebám a hlavně možnostem jak po stránce ekonomické, tak po stránce praktického využití.

[1] Může se jednat o organizaci o počtu zaměstnanců lišících se v řádek tisíců, o společnost, která může spadat do jakéhokoliv odvětví přes všechny sektory služeb (od rybolovu přes stavebnictví až ke službám).

Analýzou a návrhem prototypu aplikace začíná kapitola 7. V úvodu jsou vyjádřeny požadavky na programovou podporu posuzování softwarových procesů, dále neformální specifikace a analýza požadavků. Návrh systému obsahuje několik podrobných diagramů s popisy, a u těch, které jsou nejvíce spjaty s „business“ modely (diagramy případů použití), si systém popíšeme ještě detailněji. Je tak učiněno jako demonstrace možnosti vizuálního znázornění požadavků zákazníka, čímž se samozřejmě tyto požadavky stávají mnohem konkrétnější a ve výsledku zajišťují úroveň kvality softwarového produktu již ve fázi návrhu.

Implementační fází se zabývá kapitola číslo 8. Je zde popsána logika aplikace, popis a řešení vybraných problémů při řešení aplikace, popis vybraných částí zdrojového kódu aplikace buď důležitých z hlediska implementace požadavků, nebo zajímavých z pohledu způsobů a stylu metod a funkcí vývojové technologie, která byla vybrána v kapitole 6.3. Dále obsahuje instalační instrukce a technické detaily prostředí, ve kterém se aplikace testovala.

Závěrečné shrnutí reflektuje dosažené výsledky a jejich teoretické i praktické využití dále.

2 Management jakosti

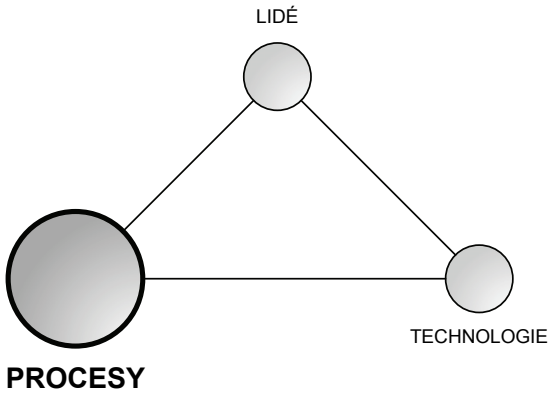
Management jakosti není zaměřen pouze na kvalitu produktu, ale také na to, jak kvalitního produktu dosáhnout. Další způsob, jak bychom management jakosti charakterizovat by mohl být následující: Jde o koordinované činnosti pro usměrňování a řízení organizace s ohledem na jakost. Zahrnuje stanovení politiky jakosti a cílů jakosti, plánování jakosti, řízení jakosti, prokazování jakosti, a zlepšování jakosti [22]. Jsou různé metody a přístupy, jak management jakosti udržet, zajistit nebo zdokonalit.

Zastánci každého z přístupů hledali jak zdokonalení, tak aplikaci na firemní procesy, pro které nebyly primárně určeny. Například metoda Six Sigma (viz 3.3) byla původně navržena pro výrobu, ale pole působnosti se rozšířilo až do sféry podnikové.

2.1 Proces, jakost, kvalita

Obecně je proces série kroků, které pomáhají řešit nějaký problém. Procesy musí být definovány jasně (nevíceznačně), což znamená, že budou používány jedním konzistentním způsobem.

Softwarové procesy jsou součástí business procesů, výsledkem SW procesů je software, u business procesů je to business. Jakost a kvalita i přes svůj hlavní význam zdaleka neznamenají totéž. V marketingové branži je jakost brána ve spojení se systémy jakosti, o kterých se budeme bavit v následující kapitole (3).



Obrázek 2-1: Tři základní faktory ovlivňující kvalitu

„Trojimperativ“ (Obrázek 2-1) znázorňuje tři základní prvky, jsou to technologie, lidé a procesy. Technologie a lidé jsou dle vyjádření na jednoduchém diagramu tím méně stěžejním ve významu vyspělosti a kvality. Úspěšné řízení znamená řízení lidských zdrojů, technologií a hlavně procesů. Bude-li proces probíhat dokonale, může být zároveň očekáván i dokonalý produkt. V procesech se produkt nejen realizuje, ale i plánuje, vyvíjí, hodnotí a zlepšuje [22].

Kvalitu je možné definovat dle různorodých přístupů, některé z nich jsou zmíněny v [22]:

- *Kvalita je způsobilost pro užití (Juran).*
- *Kvalita je shoda s požadavky (Crosby).*
- *Kvalita je to, co za ni považuje zákazník (Feigenbaum).*
- *Kvalita je minimum ztrát které výrobek od okamžiku své expedice dále společnosti způsobí (Taguchi).*

- *Kvalita je míra výsledku, která může být kategorizována v různých třídách.*

Dále se budeme zabývat kvalitativními charakteristikami SW produktů.

2.2 Kvalitativní charakteristiky softwarových produktů

Během vývoje, testování a zavedení do provozu se u softwarových systémů a procesů zkoumají různé vlastnosti. U některých vlastností mohou vznikat průniky množin jejich zájmu. Je to logické, neboť slova jsou víceznačná a je potřeba si uvědomit, co je zásadní pro dané měření. V podstatě žádná z nich není exaktně měřitelná a je na nás jaký zvolíme model či postup pro získání ohodnocení té dané charakteristiky a jaký význam pro nás bude mít dosažení určitého počtu bodů (procentuelní úspěšnosti) v dané oblasti.

Následuje výpis poměrně rozsáhlého množství charakteristik [8], [9] a [13]:

- Funkčnost (Functionality), která obsahuje Přiměřenost (Suitability), Přesnost (Accuracy), Schopnost spolupráce, Interoperabilita (Interoperability), Bezpečnost (Security), Shoda funkčnosti (Compliance)

A dále:

- Správnost (Correctness), Spolehlivost (Reliability - Dependability), Robustnost (Robustness), Výkon (Performance), Použitelnost (Usability), Srozumitelnost (Understandability), Udržovatelnost (Maintainability), Rozšiřitelnost (Scalability), Znovupoužitelnost (Reusability), Přenositelnost (Portability), Produktivita (Productivity), Včasnost (Timeliness), Viditelnost (Visibility), Schopnost výkonu (Efficiency), Kompletnost (Completness), Stručnost (Conciseness), Konzistence (Consistency)

Není možné využít všechny charakteristiky najednou a je důležité, aby se vybraly ty správné pro daný případ. Popíšeme si je a ty důležité (z pohledu webových aplikací) rozebereme dále [8], [9] a [13].

Funkčnost je schopnost softwarového produktu obsahovat funkce, které zabezpečují předpokládané nebo stanovené potřeby uživatele při používání produktu za stanovených podmínek. Zkoumá se splnění požadavků definovaných v zadání, poměry shod funkcí s definovanými protokoly, datovými formáty apod. Dále obsahuje funkce vycházejících z potřeb.

Přiměřenost zjišťuje, zda produkt má schopnost poskytnout funkce pro zajištění úloh a potřeb uživatele.

Přesnost vychází z technologických omezení počítačů (délka registrů), softwaru (typ kódování čísel) ale jde hlavně o vlastnost algoritmů. Dochází k zaokrouhlování a měří se.

Správnost je sada formálních vlastností softwaru, které předepisují funkční ekvivalenci mezi softwarovým produktem a funkcionální specifikací.

Udržovatelnost není to samé jako udržovatelnost u jiného produktového inženýrství. Softwarový produkt se nechová jako klasický produkt ve smyslu vystavování povětrnostním živlům a jiným fyzikálním zákonům. Udržovatelnost je všeobsahující koncepce a míní se tím jedna z následujících třech úloh:

1. opravení softwarových chyb, vad a nedostatků
2. přizpůsobení softwaru novým požadavkům
3. zdokonalení softwaru kvůli zlepšení kvalit

Pokud vezmeme pravý význam slova „udržovatelnost“, pak se jedná o využití prvního bodu a jde o *opravitelnost* (repairability). Tedy náročnost oprav produktu, vyhledání chyb ve zdrojovém kódu apod.

Dále sem patří: *analyzovatelnost, modifikovatelnost, stabilita, testovatelnost.*

Přenositelnost značí schopnost nasadit software na různé hardwarové/softwarové platformy bez nutnosti úprav. Tato charakteristika má přímý vliv na ekonomickou stránku produktu.

Patří sem také: *přizpůsobitelnost, instalovatelnost, slučitelnost, nahraditelnost, efektivnost, výkonnost, zabezpečení, uspokojení.*

Jeff Offut se ve své práci z roku 2004 [15] zabývá kvalitativními atributy webových aplikací, jako tři nejdůležitější vybral *Reliability (spolehlivost), Usability (použitelnost) a Security (bezpečnost)*. Nutno podotknout, že i v dnešní době se situace příliš nezměnila a výběr by se dal provést hodně podobně. Své poznatky k těmto charakteristikám popisuje následovně (i s využití názorných příkladů):

Spolehlivost – Úspěšnost mnoha komerčních projektů závisí na webových aplikacích, potom, když je aplikace nespolehlivá, logicky i obchod trpí a není úspěšný. Základna uživatelů webových aplikací je obrovská a chtějí například nakupovat na internetu v aplikacích, které ve výsledku vykazují stejnou spolehlivost jako kdyby šli do zelinářství pro okurky. Pokud „něco“ nesedí, jednoduše mohou přejít na jinou stránku a na tu Vaši se již nevrátit. Ztráta zákazníků opět logicky implikuje ztrátu peněz. Proto společnosti, které chtějí dělat „web-business“ musí vynaložit nemalé množství prostředků k zajištění vysoké spolehlivosti. Vlastně, nemohou si dovolit tuto spolehlivost nezajistit.

Použitelnost – Použijeme-li opět předchozího příkladu o nákupu okurek, tak i v tomto případě chce zákazník vykonávat webové transakce (vložení do košíku, nákup) stejně jednoduše jako v klasickém obchodě. K tomuto tématu existuje množství studií jak ze strany samotné použitelnosti, tak ze strany designu uživatelského rozhraní, které bezpochyby k použitelnosti patří. A opět platí: pokud se zákazníkovi bude zdát nějaká stránka nedostatečně vyhovující z hlediska použitelnosti, není nic jednoduššího, než přejít jinam.

Bezpečnost – O ztrátě soukromých informací uložených při registraci zákazníka již slyšel každý. Je to jen jeden z příkladů potenciální bezpečnostní chyby webových aplikací. Pokud je webová aplikace zaměřena na distribuci online brožur, tak jsou bezpečnostní následky malé. Dnes, pokud se společnosti přihodí újma způsobená bezpečnostní trhlinou v systému, tak i po nákladné nápravě propadá do dalších ztrát, neboť ztrácí zákazníky, kteří tímto utrpěli. Proto webový software musí s daty uživatele zacházet tak bezpečně, jak to jen jde. Softwarová bezpečnost je jednou z nejrychleji rostoucích odvětví výzkumu v oblasti počítačové vědy, ale vývojáři webových aplikací čelí velkému nedostatku jak vzdělaného, tak zkušeného personálu.

Mimo tři výše uvedené charakteristiky Offut uvádí ještě 4 další: *Availability (dostupnost), Scalability (rozšiřitelnost), Maintaiability (udržovatelnost), Time to market (TTM):*

Dostupnost – Jednoduše řečeno, a opět poukázáno na příkladu zelinářství, u online zelinářství je očekávána neustálá dostupnost tzv. 24/7/365 (hodin denně/dní v týdnu/dní v roce). Nedostupná služba jakoby nebyla a opět společnost přichází o zisk.

Rozšiřitelnost – Příklad: pokud Váš online obchod vyhovuje 100 zákazníkům, určitě již nebude dostačovat, pokud jejich počet stoupne na 1 000, nebo klidně i na 10 000. Rozšiřitelnost je velice důležité nejen v tomto případě, pokud chceme obchod skutečně „rozšířit“, ale i když chceme aplikaci přidat specifické funkcionality. Na tento problém je potřeba myslet již při návrhu a co v požadavcích není zmíněno teď, může se objevit později a náklady na zavedení požadavků budou vyšší, než když by v aplikaci byl prostor pro další rozšíření.

Udržovatelnost – Tento termín jsme si popsali o několik odstavců výše. U webových aplikací je tato charakteristika chápána jako schopnost (a možnost) u systému reagovat rychle na potřeby z různých oblastí potřeb. Jakékoliv záplaty, updaty apod. by měly být dostupné a hlavně okamžitě instalovatelné. Místo délky cyklu údržby počítané na měsíce, je u webových systémů nutnost tuto dobu zkrátit na dny, nebo ještě lépe na hodiny.

Time to market – Je čas od započetí práce na systému do okamžiku, kdy je systém dostupný, uvolněn k prodeji, nasazen u zákazníka. Po lehkém prozkoumání produktů na trhu s webovými aplikacemi, často zjistíme, že není ani tak důležité, jak dobrý produkt je (což je těžko měřitelná charakteristika), ale jak rychle je uveden na trh. Ten, kdo vystihne pointu poptávky a rychle zrealizuje aplikaci, která pokryje potřeby trhu (a může se jednat o cokoliv, od jednoduchých aplikací specificky určených, nebo i staronových aplikací, které využívají funkcionality, které byly dříve nedostupné), tak vyhrává. Tento boom je například vidět na aplikacích pro iPhone/iPod distribuovaných přes App Store[2] společnosti Apple.

V této kapitole jsme provedli úvod do oblasti managementu jakosti a poměrně podrobně si rozepsali typy kvalitativních charakteristik zkoumaných u softwarových produktů. Navíc u sedmi stěžejních[3] byly uvedeny vhodné příklady pro názornou a „praktickou“ ukázku důležitosti vybraných charakteristik.

Kapitola následující pod názvem „Normy a metodiky pro posuzování způsobilosti softwarových procesů“ je zaměřena na zjednodušený popis ISO normy řady 9000 a modelu CMMI.

[2] Zde se vyskytují pouze aplikace schválené společností Apple. Podobný systém zavedl i Google pro zařízení Android – Android Market a Microsoft, který nechtěl zůstat „pozadu“, tak připravuje Windows Marketplace for Mobile (nutno podotknout, že jen pro systémy s označením Windows Mobile 6.5 a vyšší).

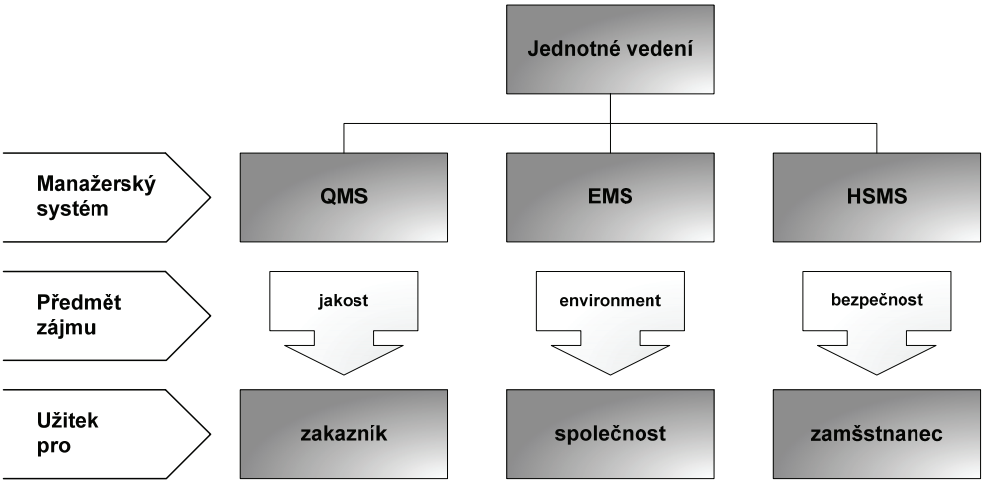
[3] Dle [15]

3 Normy a metodiky pro posuzování způsobilosti softwarových procesů

Někteří považují za kvalitní takový výrobek či službu, který je bezvadný, jiní očekávají co nejlepší parametry. V posledních letech je stále více kladen důraz na stabilitu jakosti. Tu lze zajistit jednak důslednou výstupní kontrolou (je však drahá), nebo lze kvalitu implementovat do výrobku během jeho přípravy a výroby. V tomto případě hovoříme o řízení jakosti, respektive o systému řízení jakosti (QMS - Quality Management System) [22].

V této kapitole se nebudeme do hloubky zajímat o detaily norem, ale nastíníme logiku a hierarchii uvedených systémů řízení kvality. Systém řízení kvality je zavedení pořádku ve firmě. Nejde o nic jiného, než o dobrý systém řízení.

Jak bylo výše uvedeno, naším tématem je QMS. Na obrázku Obrázek 3-1 je znázorněn QMS v kontextu s ostatními manažerskými systémy.



Obrázek 3-1: Vztahy a zaměření manažerských systémů jakosti, ochrany životního prostředí a bezpečnosti [22]

3.1 ISO 9000

ISO (International Standard Organization) 9000 je rodinou standardů pro systémy řízení jakosti. Patří zde normy řady 9001, 9002 a 9003.

3.1.1 Obsah normy ISO 9001:2000

V normě naleznete tyto kapitoly, jež definují všechny požadavky na procesní systém managementu jakosti:

1. Předmět normy
2. Normativní odkazy
3. Termíny a definice
4. Systém managementu jakosti
5. Povinnosti managementu
6. Management zdrojů
7. Realizace produktů
8. Měření, analýza a zlepšování

Normy ISO zavedly do praxe řízení jakosti některé nové přístupy [22]:

- pořádek samozřejmostí,
- respektování zákonných požadavků,
- orientace na zákazníka,
- zapojení všech pracovníků do úsilí o jakost,
- dokumentování rozhodujících provozních činností,
- identifikování klíčových procesů a zabezpečení jejich způsobilosti, monitorování a měření procesů a výrobků,
- zjišťování případných neshod a určování nápravných a preventivních opatření, vedení záznamů,
- vyhodnocování zjištěných údajů a přijímání zlepšovacích opatření.

Opět dle [22] je hlavním a výchozí zásadou orientace na zákazníka[4] (což je patrné i z obrázku Obrázek 3-1). Podstatou je poznat současné a budoucí potřeby zákazníků a plnit dodávanými výrobky a/nebo službami jejich požadavky či dokonce překonávat jejich očekávání.

Poslední vydaná česká verze mezinárodní normy ISO 9000:2008 je pod označením ČSN EN ISO 9000:2009. Verze 9000:2008 neobsahuje žádné výrazné změny oproti starší 9000:2000, kterou tím pádem nahrazuje. Poskytuje vysvětlení požadavků po osmi letech zkušeností s prováděním auditů po celém světě a zavádí změny jejichž cílem je zlepšit soulad se systémem environmentálního managementu podle ISO 14001:2004[5].

3.2 CMMI

CMMI Vychází z anglického „The Capability Maturity Model Integration“, ve volném překladu „integrovaný účinný a funkční model“ nebo „stupňovitý model zralosti“. Autorem CMMI modelu je skupina pracovníků Carnegie Mellon University a první vydání CMM modelu bylo v roce 1987. Slouží pro vývojové týmy k určení kvality organizace práce. Neudává návod nebo přesný postup, jak dosáhnout cíle, ale o doporučené postupy, jak získat odpovídající výstup pomocí kvalitního plánování a řízení prací.

Na rozdíl od norem ISO, není CMMI vhodné pro všechny odvětví. Do verze 1.2 se CMMI dělí na následující disciplíny [4]:

CMMI-SW (CMMI Software Engineering) – Softwarové inženýrství

Zaměřuje se na aplikování systematických, disciplinovaných a kvantifikovatelných přístupů k vývoji, provozu a údržbě softwarových systémů.

CMMI-SE (CMMI System Engineering) – Systémové inženýrství

Zabývá se vývojem celých systémů, může také zahrnovat SW. Systémoví inženýři se zaměřují na potřeby zákazníka a jeho očekávání. Výsledkem je řešení, kde zajišťují podporu v průběhu jeho životního cyklu.

IPPD (Integrated Product and Process Development) – Integrovaný vývoj produktů a procesů

[4] Možné termíny použité v literatuře jsou také: kvalita pro zákazníka (QFC – Quality for Customers), úplné uspokojení kvalitou (TQS – Total Quality Satisfaction) a zákazník v ohnisku zájmu (CF – Customer Focus) [22]

[5] Mezinárodní organizace pro normalizaci, Úřad pro technickou normalizaci, http:/www.wamri.cz a http://www.info-kvalita.cz/novinky/csn_en_iso_9001_2009_vydana/

Jde o systematický přístup zajišťující včasnou spolupráci s partnery a dodavateli v průběhu životního cyklu produktu. Hlavním cílem je uspokojit potřeby zákazníka.

SS (Software Subcontract) – Výběr dodavatelů a řízení objednávek

Model nabízí analýzy zdrojů a monitorování subdodávek za účelem vykonání projektových činností a úprav projektů.

Od verze 1.2 však dochází k určitým změnám.

Příklad, který reflektuje vhodnost použití předchozích disciplín je uveden v [4]:

Firma Abko staví výpočetní systémy, nakupuje COTS HW, ten modifikuje a upravuje pro něj SW. Využívá integrované týmy. Společnost může využít tyto varianty disciplín CMMI:

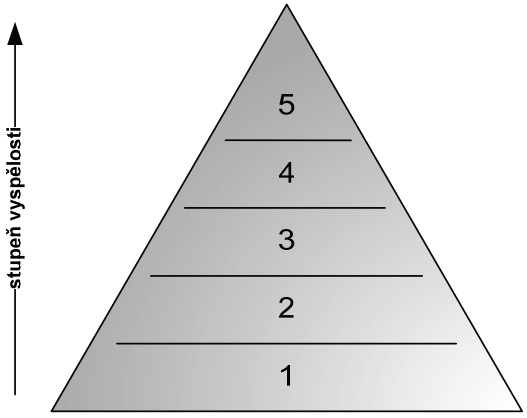
CMMI-SW, aplikace na vývoj SW,

CMMI-SE/SW, aplikace na počítačové systémy a na SW,

CMMI-SE/SW/IPPD, aplikace na systémy, na SW a na integraci týmů,

CMMI-SE/SW/IPPD/SS, aplikace na systémy, na SW, na integraci týmů a na nákup COTS HW

3.2.1 Stupňovitá reprezentace



Obrázek 3-2: Stupeň vyspělosti organizace

Obrázek 3-2 zobrazuje vyspělost organizace na základě nejvyššího dosaženého stupně. Pokud chce přejít například na čtvrtý, musí splnit tři předcházející.

Vysvětlení pojmů [2]

Úroveň vyspělosti (Maturiy Level) – Stupeň procesního zdokonalení pomocí užití předdefinovaných sad procesních oblastí, ve kterých jsou všechny cíle v sadě dosaženy.

Procesní oblasti (Process Areas) – Seskupení příbuzných praktik v oblasti, kde v případě kolektivního aplikování dosáhnou sady cílů, které jsou ustanoveny jako důležité pro zdokonalení v dané oblasti. Všechny CMMI procesní oblasti jsou shodné pro oba CMMI modely (stupňovitý i kontinuální).

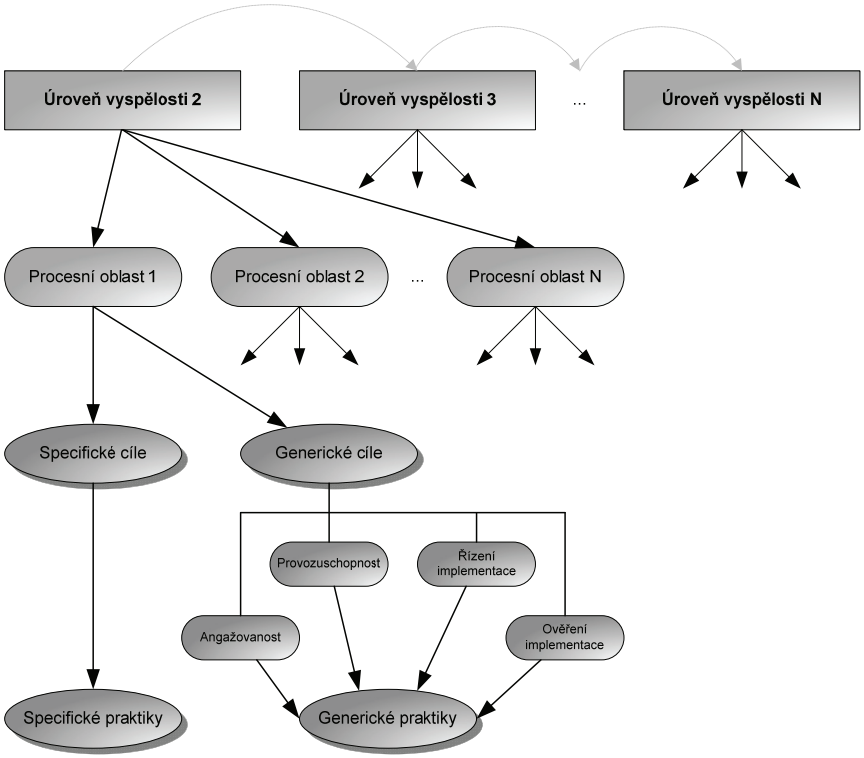
Specifické cíle (Specific Goals) – Nutný komponent modelu, který popisuje unikátní charakteristiky, které musí být přítomny ke splnění dané procesní oblasti.

Generické cíle (Generic Goals) – Nutný komponent modelu, který popisuje vlastnosti, které musí být přítomny k institucionalizaci procesů implementujících procesní oblast.

Specifické praktiky (Specific Practics) – Očekávaný komponent modelu, který je považován za důležitou k dosažení asociovaného specifického cíle. Specifické praktiky popisují aktivity plynoucí k dosažení specifického cíle procesní oblasti.

Generické praktiky (Generic Practics) – Očekávaná komponenta modelu, která je považována za důležitou k dosažení asociovaného generického cíle. Generické praktiky popisují aktivity plynoucí k dosažení generického cíle a přispívají k institucionalizaci procesů asociovaných s procesní oblastí.

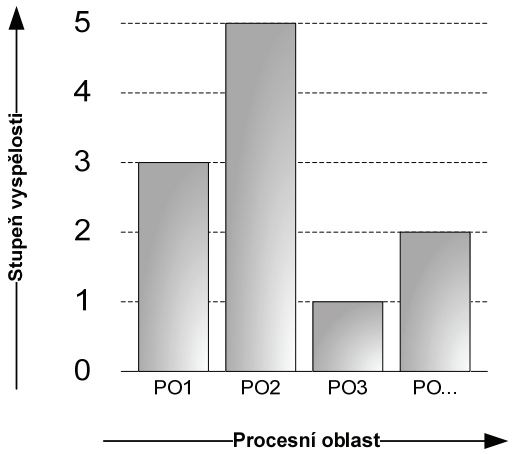
Aby organizace mohla postoupit na vyšší stupeň vyspělosti, musí splnit generické a specifické praktiky dané procesní oblasti. Celkově toto musí splnit u všech oblastí (Obrázek 3-3).



Obrázek 3-3: Stupňovitá reprezentace CMMI modelu

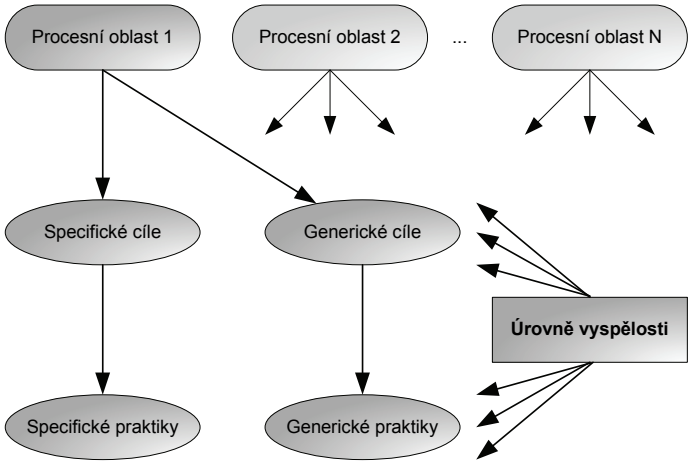
3.2.2 Kontinuální reprezentace

Na rozdíl od stupňovité reprezentace modelu, se reprezentace kontinuální zaměřuje na jednotlivé procesní oblasti a zkoumá, na jaké úrovni se právě daná procesní oblast nachází. Odpadá tak nutnost svázanosti mezi procesními oblastmi a není tolik kladen požadavek na stejnou úroveň vyspělosti v daných oblastech. Naopak je problém, pokud chce organizace vystupovat navenek s určitým dosaženým stupněm. Potom je vhodnější použití stupňovité reprezentace.



Obrázek 3-4: Příklad různých stupňů vyspělosti procesů v kontinuální reprezentaci CMMI modelu [4]

Obrázek 3-4 znázorňuje různě vyspělé procesní oblasti v organizaci a Obrázek 3-5 dále znázorňuje podmínky nutné pro postup v dané procesní oblasti. Nemusí se splnit všechny specifické a generické cíle, ale pouze ty, které patří procesní oblasti.



Obrázek 3-5: Kontinuální reprezentace CMMI modelu

Od verze 1.2 CMMI se tyto dva modely v podstatě slučují. Avšak existuje jedna architektonická vlastnost, která tyto dva modely stále rozlišuje, a tím jsou generické cíle a praktiky.

3.3 Six Sigma

Posledním systémem řízení, který si uvedeme je Six Sigma. Six sigma je strategie řízení, původně vyvinutá společností Motorola. Dnes se používá v různých odvětvích průmyslu. Six Sigma si klade za cíl identifikovat a odstranit příčiny defektů a chyb v procesech výroby a obchodu. Používá sadu metod řízení kvality, obsahující statistické metody, a vytváří speciální infrastrukturu lidí v organizaci. Každý projekt Six Sigma, vedený v organizaci, s sebou nese posloupnost kroků a má kvantifikovány finanční cíle (cenu nákladů a zisku).

Následující kapitola obsahuje popis sedmi klasických a sedmi nových metod a nástrojů řízení jakosti. Tyto nástroje jsou uplatňovány v systémech řízení jakosti, které byly uvedeny v této kapitole. Pro správné pochopení je nutná důkladná analýza potřebné normy, která by byla nad rámec této diplomové práce.

4 Metody a nástroje řízení jakosti

Není jednoduché v softwarovém vývoji definovat procesní způsobilost softwarového vývoje v číslech nebo jinak statisticky vyjádřit. Dosáhnutí statistické kontroly procesů ve vývoji zahrnuje více než tabulkování a sestavování diagramů. Zpravidla je nutné využít nových vývojových technologií, CASE nástrojů, technik odhadů spolehlivosti apod. Ačkoliv použití následujících elementárních nástrojů správným způsobem mohou mít pozitivní vliv na zdokonalení procesů a řízení kvality v softwarovém vývoji.

Tato kapitola bude rozdělena do dvou částí, kde v první si popíšeme sedm základních nástrojů řízení jakosti, a ve druhé bude sedm dalších rozšiřujících nástrojů.

4.1 Prvních sedm nástrojů řízení jakosti

Sedm základních statistických nástrojů pro podporu rozhodování v otázkách kvality procesů jsou využívány zejména v průmyslové výrobě, uplatnění však najdou v různých odvětvích řízení. Punc medializace těmto nástrojům dal roku 1989 Kaoru Ishikawa a někdy se podle tohoto matematika skupina diagramů a analýz nazývá.

Je mnoho způsobů, jak analyzovat softwarové metriky a tyto nástroje reprezentují základní operace při kontrole kvality. Nedávají však návod a ani informace vývojářům jak zlepšit kvalitu jejich návrhu nebo implementace [1].

4.1.1 Kontrolní tabulka

Kontrolní tabulky využijeme v případě jednoduchého sběru dat například s lokací, kde tato data byla generována. Je možné rychle, jednoduše a efektivně zaznamenávat informace jak kvantitativního, tak kvalitativního charakteru. Daty rozumíme zkoumané informace (závady, výskyty chyb,…). Kontrolní tabulky je možné rozdělit do následujících pěti oblastí[6]:

Klasifikace – Rys (např. vada) musí být klasifikován do určité kategorie.

Lokace – Fyzická lokace zkoumaného rysu je zavedena do obrázku části zkoumaného produktu. Využijeme zejména u průmyslových produktů.

Frekvence – Přítomnost nebo absence rysu nebo kombinace více rysů je předmětem zkoumání frekvence.

Rozdělení souboru měření – Soubor je rozdělen na intervaly, kde jsou jednotlivá měření zaznamenávána. Slouží jako podklad pro sestrojení histogramu.

Kontrolní seznam – Je jednoduchý soubor rysů. Při splnění požadavku/úkolu se daný rys zaškrtne. Podobně jako kontrola přítomnosti/absence žáka ve škole v třídní knize.

4.1.2 Vývojový diagram

Vývojový diagram je znázornění procesů (softwarových, ve firmách,…) a zároveň může být součástí dokumentace (posloupnost výpočtu, struktura algoritmu apod.).

Stavební bloky a symboly vývojového diagramu

I když by se na první pohled mohlo zdát, že jsou vývojové diagramy jednoduchou záležitostí, není tomu tak a je nutné (nejen kvůli ISO specifikaci) dodržovat základní pravidla. Mezi symboly patří:

[6] http://en.wikipedia.org/wiki/Check_sheet

Počáteční a koncový symbol je tvořen zaoblenými obdélníky (zřídka ovály) a obsahuje frázi vyjadřující „Start“ a „Konec“ (tedy např.: „Inicializace aplikace“ nebo „Ukončení aplikace“).

Šipky znázorňuji to, čemu se v počítačové vědě říká řídící struktura. Šipka jednoduše znázorňuje, že řízení přechází z výchozího symbolu (kde šipka začíná) do symbolu následujícího (kam šipka směřuje). Intuitivně jsou hlavní směry dolů a doprava. Úsečky (šipky) tvořící čáry mohou být svislé nebo vodorovné a mohou se křížit. Zároveň je vhodné šipky popsat, čímž docílíme srozumitelnosti.

Dílčí kroky jsou znázorněny obdélníky a říkají, co probíhá v algoritmu (např.: „sečti X a Y“, „ulož změny“ apod.).

Vstupy/výstupy jsou reprezentovány rovnoběžníky (např.: „přečti X“, „zobraz Y“, apod.).

Větvení postupu v algoritmu je znázorněno pomocí kosodélníku a vyjadřuje podmínku nebo rozhodnutí. Zpravidla odpovídá na otázku typu „Ano/ne“ (případně „True/False“).

Dalšími symboly mimo jiných jsou:

Dokument – obdélník se zvlněnou spodní hranou.

Manuální vstup – rovnoběžník s horní hranou stoupající zleva doprava. Např. zadání dat do formuláře.

Manuální operace – lichoběžník s delší základnou nahoře. Vyjadřuje operaci, která musí být vykonána manuálně.

Při tvorbě vývojových diagramů je nutné si klást otázky: „Co se stane nejdříve“, „co se stane dále“, „co se bude dít při chybném/správném výsledku“ a podobně.

4.1.3 Histogram

Histogram je grafická reprezentace počtu výskytů vzorku nebo populace. X-ová osa zobrazuje jednotkové intervaly parametru vzrůstající zleva doprava, Y osa obsahuje frekvenci výskytů. Na rozdíl od Paretova diagramu histogram zobrazuje sloupce podle parametrů osy X, kdežto Paretův diagram na ose X zobrazí jako první sloupec s hodnotou nejvyšší frekvence.

Účelem histogramu je znázornění distribučního rozdělení dle parametru jako celkový obraz, střední hodnota, rozptyl a koeficient šikmosti (zešikmení). Rozšiřuje porozumění zkoumaného parametru.

4.1.4 Paretův diagram

Paretova analýza je frekvenční graf sestupné tendence, kde sloupce frekvence jsou většinou spjaty s typy problémů. Tento typ grafu je pojmenován po italském ekonomovi z devatenáctého století – Vilfredo Pareto. Objasnil rozdělení bohatství, kde velké množství majetku je vlastněno malým procentem populace. Juran roku 1950 aplikoval princip k identifikaci problému kvality. Nejvíce problémů náleží malému procentu možných příčin.

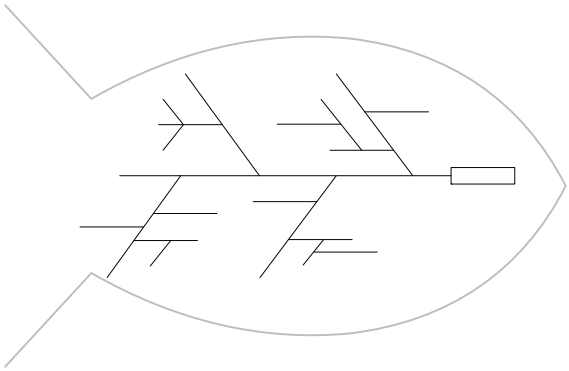
V softwarovém vývoji osa X zpravidla zobrazuje příčinu závady a osa Y počet závad. Jak již bylo řečeno u histogramu, Paretův diagram zobrazí sloupce sestupně dle množství výskytu a je tak proto možné ihned rozlišit nejvíce zásadní problémy co se týče frekvence četnosti a naopak rozpoznat ty problémy, které nejsou dle této analýzy natolik stěžejní. Dle některých postupů jsou první problémy právě ty, které by se měly řešit jako první kvůli odstranění největšího počtu závad a kvůli zdokonalení procesů vývoje softwarových produktů.

Někdy bývá Paretova analýza označována jako *princip 80-20*, protože 80% problémů je způsobeno zhruba 20 % příčin. Samozřejmě jsou tato čísla spíše teoretická a ne vždy jsou všechny případy právě v tomto rozsahu.

4.1.5 Diagram typu příčina a následek

Diagram Kaora Ishikawy – „příčina a následek“ (nebo též „rybí kost“ – dle výsledného tvaru grafu) je jednoduchým nástrojem, jehož cílem je nalezení nejpravděpodobnější příčiny řešeného problému. Je základním typem grafu v oboru „jakosti“ a žádné poradenství by se bez něj nemělo obejít (ačkoliv dle [1] je nejméně používaným typem nástroje v softwarovém vývoji).

Vhodným postupem při tvorbě Ishikawa diagramu je brainstorming. Pomůže nám nadefinovat všechny možné (i málo pravděpodobné) příčiny problému, který řešíme. Jde tedy o týmovou metodu.



Obrázek 4-1: Ishikawa diagram (příčina a následek – rybí kost)

4.1.6 Bodový diagram (regresní a korelační analýza)

Bodový diagram zobrazuje vztah dvou proměnných, kde osa X je pro *nezávislou proměnnou* a osa Y pro *proměnnou závislou*, každý bod v diagramu je výsledkem pozorování a měření závislých a nezávislých proměnných. Ovšem aplikace samotného diagramu není tak jednoduchá. Často je nutné použít jiné techniky jako korelační či regresní analýzu.

Korelační koeficient udává míru závislosti mezi dvěma proměnnými X a Y. Je v rozsahu -1 až +1. Jestliže je koeficient +1, pak je mezi oběma veličinami silná přímá závislost a naopak při -1 je závislost silná nepřímá. Při koeficientu rovno nule jsou veličiny nezávislé. Tento postup měření stochastické těsnosti se nazývá *korelační analýza*.

Pomocí *regresní analýzy* odhadujeme hodnotu náhodné veličiny (závislé proměnné) na základě znalosti jiných veličin (nezávislé proměnné, kovariát,…). Výsledkem může být regresní přímka.

4.1.7 Regulační diagram

Dle [11]: „Předmětem statistického řízení výrobního procesu (SPC – Statistical process control) je napomáhat k dosažení a udržení výrobního procesu na přípustné a stabilní úrovni tak, aby byla zajištěna shoda produktů a služeb se specifikovanými požadavky.“ Jinak bychom mohli této definici rozumět tak, že regulační diagram je v SPC nástrojem k určení, zda výrobní, business nebo softwarové procesy jsou ve stavu statistického řízení nebo ne. Jestliže jsme si na základě průběhu grafu jisti, že je proces pod kontrolou, můžeme jej s určitou mírou jistoty použít k předpovědi dalšího průběhu procesu. Pokud nastane opačný případ, zobrazený typ modelu nám může pomoci najít zdroj odchylky a eliminací docílíme návratu k normálnímu průběhu procesu.

Systém příčin, který udává stav procesu ve smyslu dalšího průběhu, je buď:

Systém náhodných příčin. Jde o *statisticky zvládnutý proces* (predikovatelný). Tyto vlivy působí v malém rozsahu, a i když je jich velký počet, žádný z nich nepřevyšuje ostatní a tím má proces stabilní rozdělení podobnosti.

Systém zvláštních příčin (vymezitelné příčiny) vyvolává nepredikovatelné změny v procesu.

Proces je stabilní (pod kontrolou), pokud sledované veličiny odpovídají požadavkům na stanovená kritéria. Hodnoty veličin musí být při pravidelné kontrole v intervalu určeném horní a spodní mezí.

4.2 Sedm nových nástrojů řízení jakosti (nástroje managementu a plánování)

4.2.1 Diagram relací

V mnoha problematických situacích vznikají složité vztahy mezi různými prvky problému (mezi procesy produktu) a často je není možné uspořádat do klasických hierarchických struktur nebo matic. Diagram relací pomáhá řešit tuto situaci tak, že jednotlivé problémy (procesy) popíše pomocí sítě obdélníků a šipek. Obdélníky vyjadřují problémy (procesy) případně úkoly problému (podprocesy) a šipky orientované vztahy mezi jednotlivými obdélníky[7].

Nejčastějším využitím diagramu relací je znázornění vztahů mezi problémy/procesy a jejich příčinami. Stejně tak může být rozebrán složitější problém/proces a ukázat návaznost a vztahy mezi prvky problému.

4.2.2 Afinitní diagram

Tento nástroj organizuje velké množství zdánlivě nesouvisejících dat a informací za účelem zatřídění do skupin na základě přirozených vztahů. Takové rozčlenění umožní dobrý nadhled nad celým problémem a umožní lépe konkretizovat hlavní příčiny, které je možné dále rozvíjet a upřesňovat.

Tvorba diagramu může být provedena několika způsoby. Buď pomocí specializovaného softwaru, nebo použitím tužky a papíru anebo je možné využít klasických samolepících poznámkových papírků. Z tohoto postupu také vznikl smysl a vzhled afinitního diagramu. Stavba afinitního diagramu je také známa jako „tvoření KJ“. KJ jsou iniciály japonského tvůrce a zakladatele – Kawakito Jiro.

4.2.3 Diagram typu strom

Abychom lépe porozuměli problému, je nutné jej rozložit na dílčí části. Na počátku vytváření diagramu si stanovíme nosné téma, tím může být například: problém, stanovený cíl, dosažený stav, požadavek. Téma dále dělíme na kategorie a jejich dílčí prvky. Prakticky se rozebírá abstraktní/netransparentní téma na jednodušší části a získáváme tak jeho specifika a konkrétní podčásti. Samozřejmě lze takto rozebrat procesy na jejich subprocesy a tím také docílit nalezení případného problému.

Stromový diagram se dá využít různými způsoby a dává k dispozici vysoce srozumitelnou a přehlednou strukturu informací. Vhodný je také při převedení požadavků zákazníků do znaků jakosti produktu.

4.2.4 Maticový diagram

Stejně jako diagram afinity a diagram příčin a následků nám maticový diagram umožňuje řešit vztahy mezi prvky (části problémů). Maticový diagram navíc zvládá řešit vztah i mezi rovinami navzájem, což prvně dvěma zmíněné diagramy neumožňují a jsme vázáni na využití jedné roviny. Systematické

[7] http://syque.com/improvement/Relations%20Diagram.htm

vymezení potřebných vztahů včetně jejich kvantifikace nám poskytuje lepší možnosti pro vyhodnocení, a tím pádem efektivnější rozhodování o budoucím vývoji [12].

Roviny jsou tvořeny danou skupinou charakteristik a její vzájemné uspořádání tvoří *matice znaků*. Matice může být tvořena parametry procesu, různými faktory, vlastnostmi produktu apod. Propojením vzniká *matice vztahů*. Možné způsoby značení jsou v následující tabulce [Tabulka 4-1].

Příklady vztahů			
● silný	■ velmi dobrý	+ + + velmi spokojen	o odpovídá
◎ střední	◘ dobrý	+ spíše spokojen	s spolupracuje
• slabý	○ uspokojivý	- spíše nespokojen	i je informován
	▫ neuspokojivý	- - - velmi nespokojen	
	▪ zcela nevyhovující		

Tabulka 4-1: Příklady vztahů v maticovém diagramu [12]

Podle metod porovnávání mezi seznamy rozlišujeme tyto typy maticových diagramů (zpravidla se označují dle písmen, která připomínají vzhledem) [12] a[8]:

Matice tvaru střecha – porovnáváme vztahy mezi prvky jedné roviny (seznamu).

Matice tvaru L – porovnáváme jeden seznam prvků proti druhému. Je to nejtypičtější využití maticového diagramu.

Matice tvaru T – zde se porovnávají vztahy mezi třemi rovinami tak, že se prvky jedné roviny srovnávají s prvky dvou dalších rovin.

Matice tvaru X – používá se pro uspořádání čtyř rovin, a to vždy po dvou. Není tedy možné porovnávat všechny seznamy navzájem, to je možné až po podrobnější analýze matice.

Matice tvaru C – porovnáváme tři roviny mezi sebou, všechny zároveň.

Matice Tvaru Y – opět se porovnávají tři roviny, ale po dvou mezi sebou.

4.2.5 Analýza údajů v matici

Tento nástroj slouží k vyhodnocování souvislostí dat uvnitř jedné roviny a mezi všemi rovinami navzájem, což maticový diagram umožňoval vždy pouze pro bilaterální vztahy. Použitím vhodných nástrojů je možné odkrýt na první pohled „neviditelné“ vztahy mezi prvky jak jedné, tak více rovin.

Jednoduchými nástroji pro zkoumání dat o vícenásobných proměnných jsou např.: *korelační diagram, plošný diagram* a *portfolio analýza.*

Korelační diagram

Nástroj pro odhalování „skrytých“ vztahů mezi prvky dvou různých dimenzí.

[8] http://syque.com/quality_tools/toolbook/Matrix/how.htm

Plošný diagram

Podobným způsobem jako korelační diagram pracuje i plošný diagram. Rozdíl je v tom, že všechny měřené hodnoty jsou srovnávány se zvoleným *optimem*. Optimum může mít různé hodnoty a zpravidla bývá vyjádřen nejvyšší možnou dosaženou hodnotou. Někdy je ale samozřejmě žádoucí minimum, průměr nebo jiná hodnota.

Osy grafu tvoří paprsky dle znaků, které zaznamenáváme, na osách je zároveň vyznačeno optimum. Na osy (paprsky) se nanesou hodnoty znaků a spojením hodnot vznikne plocha, která svou polohou vypovídá o vztahu k optimu.

Portfolio analýza

Jiným označením tohoto nástroje je „Bostonská matice“. Informace o prvcích jedné matice znaků jsou uspořádány na základě volby opět dvou hodnotících kritérií do přehledné formy, která zprostředkuje nové pohledy pro následnou analýzu a rozhodování [12].

4.2.6 Rozhodovací diagram (PDPC – Process Decision Program Chart)

Osvědčeným způsobem plánování je rozložení procesů/úkolů na hierarchii pomocí stromového diagramu. PDPC stromový diagram rozšiřuje ve smyslu identifikace rizik, selhání, nebezpečí apod. a jejich dopadu na proces. Stromový diagram je rozšířen o několik úrovní a PDPC je aplikován na spodní úrovně procesu za účelem determinace krizových protiopatření v případě možného výskytu selhání, chyby atd (pro zvýraznění některých typů problémů se používají různě tvarované boxy).

PDPC je podobné vysoce frekventované metodě FMEA (Failure Modes and Effects Analysis), oba modely identifikují rizika, důsledky chyb a související akce s tím spjaté. FMEA je metodou analýzy projevů a důsledků poruch, patří mezi metody analýzy spolehlivosti systému a je univerzálně využitelná jak v produktovém managementu, tak v řízení procesů.

4.2.7 Síťový diagram aktivit

Jedná se o klasický nástroj projektového plánování, který na rozdíl od ostatních nástrojů „dává“ dohromady části složitějších aktivit, kde probíhá více úkolů nejen sériově, ale zejména paralelně. Umožňuje nám propočítat, který úkol začne jako první, který následuje atp., dále umožní uspořádat jednotlivé aktivity procesu do logické posloupnosti a následně i do časového vymezení (doby trvání činností a tím pádem i celého projektu). Co je však důležité, že umožňuje identifikovat časově kritické cesty a vyzdvihnout místa, kde mohou být časové rezervy. Výsledek je východiskem pro konstrukci *Ganttova časového diagramu*. Síťový diagram aktivit je vhodné využít jak u malých, tak i větších projektů.

Síťové diagramy umožňují následující:

- Určit celkovou dobu trvání celého projektu.
- Stanovit pořadí činností a vztahy mezi dílčími úkony projektu (návaznost, souběžnost,…).
- Vytvořit časový harmonogram řešení.
- Najít kritické cesty.
- Najít cesty pro urychlení projektů.
- Být dostačujícím podkladem k vytvoření Ganttových diagramů.

Následující charakteristiky jsou využívány k identifikaci časových souvislostí:

- *Doba trvání činnosti*. Ta je buď známa a je uvažováno s konkrétní časovou hodnotou – deterministicky (CPM, MPM) anebo ji lze odhadnout s určitou pravděpodobností – stochasticky (PERT).
- *Nejdříve možný začátek činnosti*. Je to okamžik, kdy může činnost započít. U vstupní činnosti je nulový, u následujících je doba součtem všech dob činností, které do uzlu vstupují.
- *Nejpozději přípustný začátek činnosti*. Je to nejpozdější okamžik započetí činnosti, aby nedošlo k časovému skluzu realizace projektu.
- *Nejdříve možný konec činnosti*. Okamžik, kdy může činnost nejdříve skončit.
- *Nejpozději přípustný konec činnosti*. Nejzazší doba, kdy musí činnost skončit, aby opět nedošlo k časovému skluzu.

Metoda CPM (Critical Path Method)

CPM je metodou kritické cesty a patří mezi nejstarší metody technik síťového plánování.

Metoda PERT (Program Evaluation and Review Technique)

Pokud není možné deterministicky stanovit délku trvání činností, protože nebyly například nikdy prováděny, využijeme metodu PERT. Protože nedokáže dobu trvání činnosti odhadnout přesně ale pouze s určitou pravděpodobností, zvažujeme tři časové možnosti pro každou činnost:

a) *optimistický odhad* (nejnižší možný dosažitelný čas, zároveň činnost nemůže skončit v kratší době)
b) *modální odhad* (nejčastěji dosahovaný čas)
c) *pesimistický odhad* (opak od *a*, maximální čas a za obzvláště nepříznivých podmínek může být překročen i tento čas)

Střední doba trvání činnosti je vypočítána z odhadů předchozích tří časových možností:

$$T = \frac{a + 4 \cdot b + c}{6}$$

Hodnota rozptylu:

$$R = \left(\frac{c - a}{6} \right)^2$$

Pokud známe všechny hodnoty *a, b* a *c*, můžeme využít PERT model.

5 Fáze a modely životního cyklu

Význačnou fází softwarového inženýrství je životní cyklus softwaru (systému). V následujících dvou kapitolách 5.1 a 5.2 rozebereme klasické fáze životního cyklu a modely, které tyto fáze aplikují.

V této kapitole se budeme opírat zejména o zdroje [8] a [13].

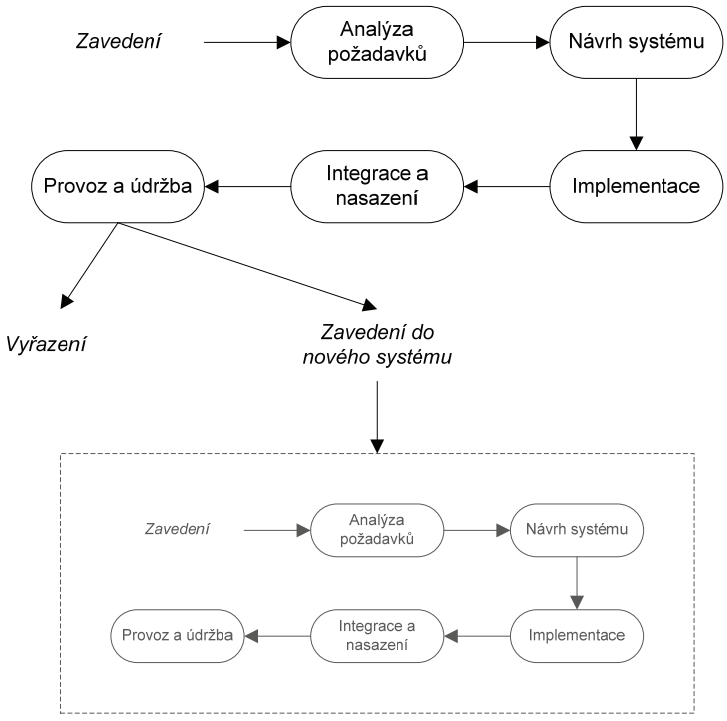
5.1 Fáze životního cyklu

Životní cyklus softwaru je abstraktní reprezentací softwarových procesů. Definuje fáze, kroky, aktivity, metody, nástroje a očekávané provedení vyvíjeného projektu. V kapitole 5.2 se dále budeme zabývat modely životního cyklu, jejich popisy a výhodami či nevýhodami oproti jiným modelům.

Pojem životní cyklus označuje změny, které nastávají během života živočichů a rostlin. Ve vztahu k softwarovým produktům se jedná o změny v průběhu „života“ softwaru. To, co je mezi „zrozením“ a „zánikem“ softwarového produktu se nazývají fáze životního cyklu. Předchází jim tzv. *zavedení (phase in)* a je zakončeno tzv. *vyřazením (phase out)*. Nejedná se o jednorázové, nýbrž postupné akce a můžeme říci, že mimo fázi *nasazení* se softwarový produkt vždy nachází právě buď v období zavedení, nebo vyřazení.

Fáze životního cyklu tedy jsou:

1. Analýza požadavků
2. Návrh systému
3. Implementace
4. Integrace a nasazení
5. Provoz a údržba



Obrázek 5-1: Fáze životního cyklu softwaru [8]

Na obrázku Obrázek 5-1 je demonstrován životní cyklus s jeho fázemi a s obdobími zavedení a vyřazení. Je vidět, že období vyřazení může vést k absolutnímu ukončení nasazeného systému anebo může vést k přechodu na systém nový, což je běžnější praxí. Rovněž je možné nechat běžet paralelně starý a nový systém, dokud se neodladí všechny potřebné požadavky na systému novém.

Dalším prvkem, který nesmí chybět v žádné fázi životního cyklu softwaru, je *testování*. Je totiž součástí každé fáze a neuvádí se (jako dříve) jako samostatná fáze.

Nyní si jednotlivé fáze popíšeme.

5.1.1 Analýza požadavků

Jedná se o aktivity směřujících k determinaci a specifikaci požadavků zákazníka na výsledný softwarový produkt. Jde o nejtěžší fázi jakéhokoliv vývojového životního cyklu softwaru, neboť komunikace probíhá mezi zákazníkem/uživatelem a vývojářem a obtížnost vyplývá z komplikovanosti komunikace jako takové. Řeší se mnohovýznamnost a nejednoznačnost požadavků. Často má zákazník problém sdělit svou představu o funkcionalitách produktu, trvá na nepodstatných součástech nebo si v některých specifikacích protiřečí či je vzájemně vyvrací.

Je mnoho metod a technik k získání požadavků, zahrnují například [8]:

- *Interview s uživateli*
- *Dotazníky uživatelům*
- *Poznámky uživatelů provádějících požadované funkcionality*
- *Studium existujících systémových dokumentů*
- *Studium podobných softwarových systémů k naučení a poznání znalostí z dané oblasti*
- *Prototypování modelů řešení za účelem nalezení a potvrzení požadavků*
- *Spojení sezení při vývoji aplikace mezi vývojáři a uživateli*

Proto se pro upřesnění požadavků využívají modely, grafy a diagramy. Standardní praxí je v současnosti využití UML, což je standardní modelovací jazyk pro specifikaci požadavků (stejně tak i pro návrh systému). Požadavky jsou specifikovány jak v grafické tak v textové podobě, modely se často překrývají, aby postihly komplexní problematiku systému.

Výstup každého životního cyklu by měl být zkontrolován a otestován. *SQA (Software Quality Assurance)* je profesionálním přístupem k testování požadavků.

5.1.2 Návrh systému

Návrh systému je popis struktur implementovaného softwaru, dat, které jsou částí systému, rozhraní mezi komponenty systému a někdy i použitých algoritmů. Ve fázi návrhu u tzv. „enterprise“ (informačních) systémů se datové struktury vztahují k databázi a tedy k návrhu databázového schématu. Většina algoritmů není plně popsána a spadá tak do volného „návrhu“ programátorů během implementační fáze.

Analýza a návrh systému se často překrývají, vedou k tomu následující dva důvody [8]:

1. Moderní životní cykly jsou iterativní s inkrementy. V jednu chvíli je tak v cyklu několik verzí. Některé v analýze, jiné v návrhu, další jsou implementovány a nakonec jsou některé již nasazeny.
2. Podstatným důvodem „mazání“ pevné hranice mezi analýzou a návrhem je použití modelovacího jazyka UML. Ten je totiž používán jak pro analýzu, tak pro návrh.

Od modelu analýzy k modelu návrhu tak vedou jemné dodatky, které specifikaci dodávají podobu více vhodnou pro návrh systému.

Do této chvíle se jednalo o tzv. *detailní návrh*, který přidává detaily modelu analýzy. Do návrhu systému dále spadá *architektonický návrh*. Je spjat s nastavením architektonické struktury systému, struktury komponent, kde musí respektovat detailní návrh, a dále se zabývá principy a vzory vzájemné komunikace mezi komponenty. Hlavním úkolem architektonického návrhu je vznik systému, který je anglickým termínem označován jako *„supportable“* (únosný, snesitelný), tedy srozumitelný, udržovatelný a rozšiřitelný (o těchto kvalitativních charakteristikách softwarového systému jsme se zmiňovali v kapitole 2.2).

Testování architektonického návrhu demonstruje, že struktura postihuje komplexnost navrhovaného systému, zajišťuje „snesitelnost“, urychluje vývoj, atp. Dále je testování architektonického návrhu potřebné kvůli verifikaci, jestli návrh komponent odpovídá principům a vzorům, které jsou vyjádřeny architektonickým návrhem.

5.1.3 Implementace

Pojem implementace je zpravidla programování, ne však přesné přepisování specifikace do programového kódu. V jistém slova smyslu je i nyní programátor návrhářem. Neboť některé oblasti návrhu nebyly postihnuty do té míry, aby je programátor jednoduše napsal. Týká se to zpravidla oblasti obsahující komplexní algoritmy. Je proto vyžadována znalost programovacího jazyka a schopnost použít správné, nejlépe osvědčené, konstrukce. Téměř nikdy se nezačíná, jak se říká, od nuly. V dnešní době je mnoho systému sestaveno z modulů, jakýchsi celistvých bloků, které splňují specifickou funkcionalitu.

Práci programátorům „zjednodušují“ (urychlují) IDE (Integrated Development Environment) – programové prostředí určené k vývoji a CASE (Computer Aided Software Engeneering) nástroje. Velká část kódu je pomocí těchto nástrojů vytvořena a programátor dopisuje zbývající „kousky“, které jsou však tím stěžejním a z hlediska fungování a provozu tou nejpodstatnější součástí programu. Nezřídka se stává, že zásahy programátora do architektury systému jsou natolik zásadní, že je nutné aktualizovat nazpět návrhový model. Tento proces se nazývá *reverse engineering*. Spolu s *forward engineering* (což je klasický postup z návrhu do implementace) se tento cyklus nazývá *roundtrip engineering*.

Důležitou součástí implementace je testování a ladění.

Ladění („debuggování“) je akt odstranění softwarových chyb („bugů“). Buď jde o syntaktické nebo logické chyby, nebo chyby v kódu.

Testování jsou aktivity, které vedou k odhalení chyb. Buď jde o posouzení implementace kvality zdrojového kódu, nebo je založen na spuštění proveditelného kódu. V tomto případě rozlišujeme dvě varianty:

1. Black-box testování, kde se zkoumá funkčnost vycházející ze specifikace návrhu
2. White-box testování, kde se vychází ze znalosti zdrojového kódu.

5.1.4 Integrace a nasazení

Integrace je fází životního cyklu, kdy se aplikace kompletuje z dříve implementovaných a testovaných komponent. Nasazení u zadavatele k používání se zpravidla provádí postupně dle verzí.

Důležitou fází integrace je *integrační testování*, kde se komponenty a moduly, které již byly otestovány v předchozí fázi (implementace), implementují souběžně do celku, který představuje komplexní systém. Někdy není možné pevně rozlišit mezi testováním ve fázi implementace a integračním testováním, jako například při agilním vývoji. Metodu „velkého třesku“ (big-bang),

kdy se veškeré komponenty, které při implementačním testování fungovaly bezvadně a byly odladěny, spojí a vytvoří tak celistvou část systému, která bude fungovat, nelze brát u složitých systémů v potaz. Integrační testování lze provádět buď *shora dolů,* nebo *zdola nahoru*.

Testování *shora dolů* vychází z přidávání komponent od kořene (který je v abstraktní pozici umístěn nahoře) směrem dolů. Nezřídka však nastává případ, kdy je nutné chybějící komponenty nahradit tzv. *náhražkami (stubs).* Jde o jednoduché komponenty a není podstatná jejich plná funkcionalita, stačí jednoduchý výpis zprávy apod. Testování *zdola nahoru* postupuje opačným způsobem a chybějící komponenty, v hierarchii postavené výše, se nahrazují tzv. *drivery.*

Nasazení, jak již bylo řečeno, není jednorázovou aktivitou, ale v souladu s inkrementálním vývojem se zavádí pomocí verzí (releasů). Předtím je však software testován v reálném prostředí – *alfa testování*. Následuje *přejímací testování,* které také testuje software na náchylnost k poruchám, bezpečnost, zotavení po poruše atp., ale testování probíhá u zákazníka, v prostředí, ve kterém se systém bude provozovat. Tato fáze může obsahovat školení uživatelů, které by mělo předcházet samotnému vydání systému. Důležitá je i uživatelská dokumentace.

5.1.5 Provoz a údržba

Zahájení provozu systému je spjato s ukončením chodu předchozího systému. Často oba systému běží simultánně, pro případ, že by nový systém selhal. Počátek údržby systému je v podstatě stejný jako uvedení systému do provozu, neboť již v počátcích se mohou objevit chyby nebo vlastnosti, které chybí nebo byly požadovány v době zahájení provozu. Rozlišujeme následující typy údržby [8]:

- *Opravná* – oprava chyb a nedostatků objevených za provozu.
- *Adaptivní* – úprava systému jako odpověď na technické nebo obchodní změny.
- *Zlepšovací* – rozšíření systému přidáním nových vlastností nebo přispění ke kvalitativním změnám.

V této podkapitole jsme si uvedli fáze životního cyklu. V další části si ukážeme, jak jsou tyto fáze v jednotlivých modelech zohledňovány, na které kladen větší důraz, ale co je hlavní, jaká je filozofie modelů vzhledem k návaznosti/prolínání sousedících fází.

5.2 Modely životního cyklu

Tato podkapitola navazuje na předchozí část (Fáze životního cyklu – 5.1). Modely nepopisují „co“, ale „jak“ pracovat v životním cyklu softwaru a tak často korespondují s kulturou firmy a zkušenostmi a dovednostmi vývojového týmu, konkrétním projektem (zaměření, velikost,...), atd.

Rozlišujeme dva základní modely:

1. Klasický (vodopád se zpětnou vazbou)
    a. Vodopád
    b. Iterativní životní cykly s přírůstky
2. Iterativní s inkrementy
    a. Spirálový model
    b. IBM Rational Unified Process (RUP)
    c. Model Driven Architecture (MDA)
    d. Agilní vývoj (Feature Driven Development – FDD, Scrum, Extreme Programming – XP)

5.2.1 Vodopád

Model vodopád je tradičním modelem životního cyklu uvedeným v 70tých letech minulého století. Používal se zejména v dávkových systémech jazyka Cobol, v současné době se při hojném využití objektově orientovaného programování tento typ modelu příliš nepoužívá.

Model více méně odpovídá postupnému plnění fází životního cyklu, tak jak jsou logicky za sebou a reflektuje tak snahu systematického vývoje. Jak již bylo řečeno dříve, vznikají mezi fázemi zpětné vazby, kdy se dokument vytvořený například ve fázi implementace, musí s v rámci některých požadavků promítnout zpět do dokumentu fáze návrhu systému. Zajímavější je ale vznik variant modelu typu vodopád, kde jsou významné dva rysy. Prvním je *překrytí* životních fází cyklu, takže fáze nejsou striktně odděleny a návrháři tak mají větší možnost a druhým rysem je využití *prototypování*. Prototypování představuje možnost prezentace nehotového systému na jednoduchém programu postihující některé (důležité) požadavky v rámci dané fáze.

Model typu vodopád je v dnešní době nevyužit, ale varianta s iterativním životním cyklem s přírůstky je již použitelnější.

Iterativní životní cyklus s přírůstky

Přírůstek v názvu varianty vodopádového modelu znamená obohacení vytvářeného systému o nějakou vlastnost, rozšíření či vylepšení (část uživatelského rozhraní, zabezpečení dat,…) po každém průchodu životního cyklu – iteraci. Jde vlastně o opakované provádění modelu typu vodopád, ale po dokončení iterace je do procesu opět zapojen zákazník. Po každé iteraci vzniká tzv. *konstrukce* (někdy i *sestavení*, angl. *build*), která v každé iteraci vytváří novou verzi systému. Oproti klasickému vodopádu jsou iterace krátké (jednotky dnů, max. týdnů), což umožňuje lepší plánování a předpověď dalšího průběhu iterací.

Jako další velkou skupinu iterativních modelů životního cyklu si představíme již zmíněný spirálový model, RUP a varianty agilního vývoje MDA, FDD, SCRUM a XP.

5.2.2 Spirálový model

Ve skutečnosti je spirálový model metamodelem tvaru spirály a může obsahovat jiné modely životního cyklu. Model je tvaru spirály procházející čtyřmi kvadranty. Kvadranty jsou *plánování, analýza rizik, inženýrství* a *hodnocení zákazníkem*. Cykly začínají *plánováním*, probíhá zde počáteční sběr požadavků a projektové plánování. *Analýza rizik* posuzuje náklady a výnosy, nebezpečí/příležitosti. Dále posuzuje rozhodnutí, zda pokračovat dále v projektu a předat do fáze *inženýrství*. Zde probíhá samotný vývoj systému a výsledkem je *build* (viz předchozí kapitola Iterativní životní cyklus s přírůstky), prototyp nebo hotový produkt, který je předán *zhodnocení zákazníkem*. Pak může začít druhý cyklus.

Opakování spirálového cyklu dává modelu vysoce iterativní charakter a díky hodnocení zákazníkem vznikají systému odladěné na míru. Jak bylo řečeno, spirálový model je metamodelem pro způsob uvažování při vývoji softwaru.

5.2.3 Rational Unified Process – RUP

RUP je komerčním produktem firmy IBM (dříve Rational Corporation). RUP není jen modelem životního cyklu, obsahuje i prostředí pro vývojáře. RUP nemá striktně vymezenou strukturu, bývá znázorněn dvojdimenzionálně. Horizontální směr reprezentují: *zahájení (inspection), rozpracování (elaboration), konstrukce a přechod (transition)*. Naopak vertikální směr znázorňuje disciplíny

podílející se na životním cyklu modelování podniku (business modelling), analýza požadavků a návrh, implementace, testování, nasazení a podpůrné aktivity.

5.2.4 Architektura řízená modelem (Model Driven Architecture) – MDA

Vychází z objektově orientovaného přístupu a jedním z hlavních cílů MDA je tzv. *spustitelná (executable) specifikace* UML, tedy systému který UML model reprezentuje. MDA předpokládá vývoj software jako posloupnost transformací z formální specifikace přes méně detailní vývojové etapy ke spustitelnému programu. Každá transformace bedlivě kontroluje, zda výstupy jsou pravdivou reprezentací vstupů.

5.2.5 Agilní vývoj

Jde o prosazení nových přístupů vývoje software, smyslem je splnění rychlých změn na požadavky a podmínky (zejména u internetových projektů). Manifest organizace, která zavedla agilní vývoj Agile Alliance obsahuje následující čtyři priority:

- *Jednotlivci a interakce před procesy a nástroji*
- *Fungující software před úplnou dokumentací*
- *Spolupráce se zákazníkem před vyjednáváním kontraktu*
- *Reagování na změny před dodržováním plánu projektu.*

Z čehož vyplývá, že agilní vývoj zdůrazňuje kreativní přístup při tvorbě programu. Upřednostňuje vývojáře, programátory a spolupracující týmy před procesy, nástroji a dokumentací. Vše ostatní je druhotné.

Feature Driven Development (FDD), Scrum, Extreme programming – XP

FDD – Neboli *vývoj řízený rysy* upřednostňuje „malé“ výsledky, kde je aplikována pouze malá část funkčnosti, ovšem ta, která je důležitá pro zákazníka.

Scrum – Vývoj probíhá v krátkých iteracích, tzv. *sprintech*, seznam akcí pro danou iteraci se zaznamenává do *backlogu.* Mimo to Scrum pracuje s mnoha jinými termíny, které popisují metodiku vývoje. Mimochodem Scrum není žádným akronymem, dle www.tfd.com: „Informal a disorderly struggle“, v praxi jde o ranní schůzku o délce cca 15 minut, kde se sejdou všichni vývojáři a řeší, co se dělalo včera a co je na programu dnes.

XP – Některé principy extrémního programování jsou programování v párech, kolektivní vlastnictví nebo refaktorizace. Jediným uznávaným, nezpochybnitelným, jednoznačným a měřitelným zdrojem informací je zdrojový kód. Tým tvoří 2 až 10 programátorů, kteří jsou schopni se při vývoji vyrovnávat s měnícím se zadáním. Můžeme zde zahrnout TDD (Test Driven Development), tedy *programové řízení testy*. Jde o agilní metodu vývoje softwaru, kdy se nejprve vytvoří testy a po té až funkcionalita.

Samozřejmě do oblasti agilního vývoje patří mnoho jiných metod. Dle zkušeností vývojářů a manažerů je vždy nutné dodržovat pravidla dané metodou a nesnažit se kombinovat různé způsoby práce.

6 Nástroje a prostředky pro vývoj webových aplikací

V současné době je na výběr několik směrů, kterými se můžeme vydat při výběru vhodného nástroje pro vývoj webových aplikací. Ať už se jedná o klasický způsob spojení skriptovacího jazyka PHP ve spojení se značkovacím jazykem XHTML nebo o nějaký moderní, rapidně vyvíjený objektově orientovaný programovací jazyk, vždy stojíme na rozcestníku, kde často není poznat, kterým směrem je vhodné se ubírat.

Rozhodují požadavky na softwarový produkt (např. napojení na jiné technologie, které umožňují pouze typizované nástroje), možnosti programovacího jazyka a v neposledním případě zkušenosti programátorů a dobrý návrh aplikace. Ovšem i v takových případech není vždy vyhráno.

6.1 Technologie zaměřené na webové aplikace

Zaměříme-li se v našem výběru a možnostech na „mainstreamové" technologie a nástroje, které jsou součástí boomu kolem pojmu RIA (Rich Internet Applications), získáme několik alternativ. Termín RIA je užíván poměrně často, a to hlavně díky marketingovým náletům společností, které se ve vývoji inkriminovaných technologií angažují. Tyto produkty jsou zajímavé zejména svými schopnostmi a prostředky určenými pro úzké použití. Jak se sám autor tohoto textu později přesvědčí, vykonání i na první pohled banálních operací bude úkolem poměrně nejednoduchým. O tom ale později, kdy si u daných technologií klady a zápory probereme.

Z velkého množství různě kvalitních produktů si podrobněji probereme Flex (společnosti Adobe), Silverlight (Microsoft) a JavaFX (Sun Microsystems).

6.1.1 Flex

Flex je sadou nástrojů pro tvorbu RIA společnosti Adobe, původní nápad však vzešel ve firmě Macromedia, kterou Adobe nakonec koupila[9]. Výsledkem zkompilovaného projektu může být SWF soubor, který lze zobrazit ve většině majoritních prohlížečů (pomocí Flash Player) nebo aplikace pro AIR (může pak běžet na straně klienta jako desktopová aplikace bez nutnosti spouštět webový prohlížeč). Tato forma má své klady i svá omezení, o nichž se dočteme později.

Flex kombinuje několik přístupů vývoje softwaru. Pro grafická uživatelská rozhraní existuje MXML, což je značkovací jazyk založený na XML a Actionscript. MXML umožňuje rychle vložit formulářové prvky, boxy, které udávají rozložení komponent, rozbalovací nabídky, ale i pokročilejší elementy jako např. tabulky s možností automatického seskupování dle zvoleného parametru, programovatelné řazení, animace přechodů mezi stavy a mnoho jiného.

Naopak Actionscript vychází z ECMAScript a je vhodný pro psaní složitějších konstrukcí, algoritmů nebo i ke generování MXML prvků. Je hodně podobný JavaScriptu (kde JS vychází právě z ECMAScript). Použití značkovacího a skriptovacího jazyka nám umožňuje oddělit vzhled (UI – uživatelské rozhraní) od logiky aplikace. Není to však striktní omezení a jak již bylo řečeno, je možné oba způsoby zápisu míchat, což se i často děje.

[9] Vývoj začal již v roce 2004 právě společností Macromedia, po té, co roku 2005 společnost koupila Adobe, není již dále potřeba vlastnit licenci pro Flex Data Services. Ten přešel jako separátní produkt pod jméno LiveCycle Data Services [http://en.wikipedia.org/wiki/Adobe_Flex]

Jelikož Flash Player verze 9 nativně nepodporuje aplikace vytvořené pomocí technologie Flex, je nutné při exportu do SWF souboru přibalit i celý framework nutný pro spuštění aplikace, čímž se velikost výsledného zkompilovaného souboru zvětší. Flash Player 10 si již však dokáže uchovat tento framework v cache paměti a následky jsou zřejmé – menší aplikace, více muziky.

Vývoj

Pro vývoj je potřebný jen patřičný Flex SDK (Software Development Kit), v současné době verze 3.3 (pod kódovým označením Moxie pro 3.x), případně betaverzi 4.0 (Gumbo), a dále pro překlad použít command-line kompilátor, jež je (jako základní verze Flex SDK) open-source. Tento způsob je nicméně nepohodlný a vzhledem k času strávenému při vývoji je mnohem lepší použít IDE:

- *Flex Builder* – Komerční IDE[10], lze zajistit tzv. „for Education" verzi pro studijní účely a ta je zdarma. Flex Builder je vystaven na Eclipse a proto je možné toto IDE rozšířit o další pluginy jako SVN apod. Výhodou oproti Eclipse, ale i jiným IDE, které nejsou zaměřeny na Flex, je, že Flex Builder obsahuje designovou část pro WYSIWYG návrh uživatelského rozhraní, čímž se vývoj samozřejmě zrychluje a usnadňuje.
- *Eclipse* – Naproti Flex Builderu je Eclipse open source vývojovou platformou a není zaměřen pouze Flex, ale opět díky pluginům je možné pracovat s PHP, C++ apod. Zapomenout také můžeme na „debugování", vizuální návrh aplikace atd.
- *Jiné IDE* – V úvahu připadá FlashDevelop a zřejmě i jiné alternativy. Nelze však počítat s plným využitím všech možností, které Flex SDK skýtá.

Datové služby

Spojení s datovými službami je možné provést několika způsoby, závisí však, na jaký backend se chceme napojit. V praxi je situace poněkud problematičtější kvůli technologii Flash Player resp. AIR, jako klientské aplikace, je připojení např. k MySQL nemožné. Řešení je takové, že využijeme HTTP a data získáme z databáze za použití nějakého „prostředníka", tedy serverového skriptu. Tato „vlastnost" je však stejná pro všechny RIA technologie, AJAX nevyjímaje. Způsob, jak data získat si popíšeme na příkladu [16]:

Pokud tedy chceme například načíst seznam zákazníků z databáze, musíme udělat následující:

1. V serverové technologii typu PHP, ASP.NET nebo Java vytvoříme skript, který data načte z databáze a předá je dál. Konkrétní implementace může být v REST stylu (například požadavek http://example.com/giveme/customers vrátí sadu zákazníků jako XML soubor) nebo se může jednat o webovou službu postavenou na protokolu SOAP - na tom vcelku nezáleží, důležité je, aby se o data dalo zažádat.
2. Ve Flexu využijeme třídu HTTPService, WebService nebo RemoteObject (podle použité implementace na serveru), o data zažádáme a po jejich obdržení je zobrazíme.

Podle různých syntetických testů[11] lze lehce usoudit, že použití RemoteObject je mnohonásobně rychlejší, než použití HTTPService nebo WebService.

Další technologie, kterou si krátce představíme, je Silverlight.

[10] Zkušební 60ti denní verze je volně ke stažení

[11] např. http://www.themidnightcoders.com/blog/2007/03/flex-remoteobject-vs-webservice.html

6.1.2 Silverlight

Microsoft je největším rivalem pro společnost Adobe, ať už jsou marketingové fráze jakékoliv, MS se snaží konkurovat Flash(i), což lze brát samozřejmě jen pozitivně, programátor má z čeho vybírat a tak často i přebere.

Silverlight je proveden jako programovatelný plugin do webového prohlížeče, podporovány jsou majoritní prohlížeče a operační systémy. Projekt Moonlight je implementací pro GNU/LINUX, ovšem zatím s omezenou funkcionalitou/použitelností. Mimo klasické operační systémy budou v příštích letech[12] podporovány i OS mobilní – Windows Mobile 6 a Symbian. O Mac OS pro IPhone/IPod podpoře ze strany Microsoftu je úsměvné jen spekulovat, natož přemýšlet o reálných termínech. Na Mac OS totiž legálním způsobem nelze spustit ani Flash.

Silverlight je vývojově poněkud pozadu oproti stále srovnávané technologii Flex resp. Flash, nicméně vyniká některými vlastnostmi, které stojí minimálně za povšimnutí. Mimo klasických funkcionalit jako podpora animací, přechodů, grafických prvků a multimedií, Silverlight od verze 2.0 umožňuje psát aplikace v .NET jazycích, jelikož implementuje plnou podporu CLR (Common Language Runtime). Programátor tak není vázán na jeden programovací jazyk, ale může si vybrat ze C#, Visual Basic .Net, Delphi nebo IronPython, tím však výčet nekončí. Dalším specifikem je podpora H.264 (od verze SL 3.0), což již v běžné praxi využijeme méně. Dále podpora 3D (Perspective 3D – jako 3D transformace 2D objektů). LINQ je ze zmíněných velice užitečná komponenta pro dotazování ve stylu SQL v .Net programovacích jazycích. Otevírání a zavírání souborů je jeden z několika nedostatků, které ve verzích nižších než 3.0[13] chyběly. Jak je vidět, v některých ohledech má Silverlight před sebou ještě dlouhou cestu, jinde je o kus dál před konkurencí.

Tak jak má Flex MXML, tak Silverlight oplývá XAML. V podstatě je syntaxe hodně podobná a ve většině případů lze podobný prvek napsat oběma „XML-based“ jazyky.

Vývoj

Zde neexistuje příliš mnoho alternativ při výběru IDE. Visual Studio 2008 (VS) je jasnou volbou, zejména kvůli podpoře .NET jazyků a kvůli propracovanosti vývojového prostředí jako takového. Hlavní nevýhodou VS je to, že neobsahuje žádný WYSIWYG editor pro Silverlight. Ovšem existuje prostředí Microsoft Expression Blend, které tento nedostatek výtečně zastupuje. Rychlost je již však dalším bodem, který ke kladným nepatří.

Další alternativní IDE jsou ve vývoji, ale např. na pluginu do již zmíněného Eclipse se pracuje.

Jako poslední z trojice srovnávaných RIA technologií je JavaFX.

6.1.3 JavaFX

I když má Sun Microsystems dlouhodobé zkušenosti s úspěšným systémem Java, resp. Kompletním řešením J2EE, na poli RIA jim vhodná technologie stále chyběla, neboť Java samotná není primárně určena pro vývoj aplikací.

Řešení, které vyplní místo na trhu, je JavaFX. Mimo desktopové systémy je zaměřena i na mobilní řešení (podobně jako FlashLite pro Windows Mobile). JavaFX je zastřešující technologií pro programovací jazyk JavaFX Script (vývoj desktopových aplikací) a JavaFX Mobile (aplikace pro mobilní zařízení). V rámci analýzy vhodné technologie pro realizaci prototypu autor v tuto chvíli

[12] Dle zdrojů http://www.wmexperts.com/silverlight-mobile-rumored-windows-mobile-65 a http://silverlight.net/learn/mobile.aspx

[13] http://dotnetaddict.dotnetdevelopersjournal.com/new_silverlight3.htm

nenachází příliš mnoho inspirace pro další zkoumání JavaFX a ani Javy, neboť dle jeho názoru je tato technologie v příliš raném stadiu vývoje a vzhledem k nutnosti spuštění JRE (Java Runtime Environment) pro chod jakékoliv Java/JavaFX aplikace, je práce s produkty vytvořenými jako webová aplikace typu RIA velice nepohodlná až odstrašující.

JavaFX má před sebou ještě dlouhou cestu a vzhledem k malé penetraci mezi uživateli není možné zhlédnout dostatečné množství příkladů, které by přesvědčily k použití této platformy.

Ačkoliv výpis frameworků pro vývoj webových aplikací zdaleka není konečný, jako stručný popis a vytvoření vhledu do situace na poli RIA technologií je předchozí text této kapitoly snad dostačující. Dále se seznámíme s možnostmi uložení dat.

6.2 Uložení dat

Data, se kterými aplikace bude pracovat, je nutné vhodně uložit. Prakticky je možné volit mezi uložením na lokálním disku a klasickými relačními databázovými systémy. V možnosti uložení na lokální disk tvoří výjimku SQLite, která má také charakter relační databáze, nicméně k datům je možné přistupovat pouze lokálně.

6.2.1 Databáze

MySQL je multiplatformní databáze, kde komunikace s ní probíhá pomocí jazyka SQL. Stejně jako u jiných databází, jsou pro SQL jistá specifika. MySQL je oblíbena z mnoha důvodů, mezi ně patří: snadná implementovatelnost (schopnost běhu na operačních systémech Windows, Linux aj. ve spojení např. se servery Apache nebo IIS), cena (open-source) a dále například podpora InnoDB tabulek (podpora transakcí, cizích klíčů), „views“ atd.

PgSQL je v mnoha směrech srovnatelným databázovým systém (a v mnoha ohledech také i lepším) než MySQL. Problémem je rozšířenost a obecná povědomost o systému. Výhodou je naopak podpora uložených procedur (PL/pgSQL).

MsSQL je výborný databázový systém. Licence jsou uzpůsobeny i pro vývojáře a testování.

Oracle si zmíníme jen okrajově, spíše jako doplnění výčtu. Jeho kvality jsou nezpochybnitelné, ovšem nasazení v našem případě je zbytečné. Navíc zásadní fakt, a tím je cena licence, je opravdu mimo náš záběr.

6.2.2 Lokální disk

XML je zřejmě nejvhodnějším stylem ukládání dat, která jsou na první pohled dobře čitelná z pouhého otevření zdrojového kódu XML souboru. Je vhodný pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Je očekávána podpora různých typů kódování a hlavně podpora ze strany použité aplikace.

SQLite[14] je relační databázový systém obsažený v relativně malé knihovně napsané v C. Na rozdíl od databází založených na principu klient-server, kde je databázový server spuštěn jako samostatný proces, je SQLite pouze malá knihovna, která se přilinkuje k aplikaci a pomocí jednoduchého rozhraní ji lze začít využívat.V SQLite je implementován téměř celý standard SQL-92.

[14] http://cs.wikipedia.org/wiki/SQLite

6.3 Výběr technologie

Z uvedených charakteristik vyplývá, že opravdu není jednoduché se bez dostatečných znalostí rozhodnout pro danou technologii. I se zkušenostmi je nutné se delší čas věnovat funkcionalitám, které jsou pro navrhovanou aplikaci stěžejní a tím pádem porovnávat s možnostmi vybrané platformy. V 95 % lze totiž tutéž věc zpracovat ve všech zmíněných prostředích.

Rozhodnutí pro výběr technologie ke zpracování aplikace pro tuto diplomovou práci je částečně podmíněno vlastnostmi a schopnostmi platforem, částečně podporou pro daný jazyk/platformu a neméně i subjektivním názorem a důvěrou v produkty osvědčené společnosti. Tím je v tomto případě Adobe a jejich Flex. Ačkoliv se Adobe zaměřuje na grafické a designové aplikace, v poslední době se ubírá i jinými směry, čímž je Flex velkým důkazem. Dle názoru autora této diplomové práce jsou produkty Adobe na vysoké úrovni[15] po stránkách funkčnosti, stability, spolupráce s jinými aplikace a v neposlední řadě výborně zpracovaným uživatelským rozhraním. Právě uživatelské rozhraní je i u AIR aplikací v „defaultním" nastavení přívětivé a propracované.

Co se týče uložení dat, byl zvolen MySQL systém[16]. Hlavně díky své dostupnosti (free) a dále jako vhodnější řešení v porovnání s SQLite. Data jsou uložena centrálně a je tak k nim možný přístup odkudkoliv.

Jako poslední podstatnou část doplňující výběr technologií uvedeme vybraný framework pro prostředí Flex, a tím je Mate.

6.3.1 Mate

Mate je navržen tak, že využívá výhod MXML a běžných událostí (events). Tím, že je část frameworku definována na aplikační úrovni, stává se Mate poměrně „nevtíravým" frameworkem díky řízení událostmi. Nevtíravý je možná nevýstižným termínem, „unobtrusive" lze chápat tak, že jednotlivé komponenty vyvíjené aplikace jsou nezávislé na frameworku Mate.

Mate využívá návrhového vzoru „Dependeny Injection" (DI)[17] – „vkládání závislostí". Jedná se o specifickou formu „Inversion of Control" (IoC) – „obrácené řízení". To umožňuje uvolnit vztahy mezi jinak pevně svázanými komponentami a oproti klasickým frameworkům není problém naprogramované pohledy, funkce a třídy „recyklovat" v jiných projektech i když budou postaveny na jiném frameworku.

Centrem Frameworku Mate je EventMap[18]. Zachytává a vypouští události a tím pádem koordinuje procesy v aplikaci.

V této kapitole jsme popsali zásadní webové frameworky, které se v současnosti používají a následně vybrali kombinaci technologií pro implementaci navrhovaného systému.

[15] Jde pouze o osobní názor autora, který vychází z dlouhé zkušenosti s používáním softwarových aplikací v dané oblasti.

[16] Toto rozhodnutí nebylo nejšťastnějším vzhledem ke komplikacím, které byly způsobeny připojením k MySQL serveru. Problémem však nebyl MySQL, ale styl, jakým Flex komunikuje pomocí RemoteObject

[17] Popis návrhového vzoru například zde: http://sqdw.signaly.cz/0804/inversion-of-control-dependency

[18] Způsob, jakým komponenty v Mate kooperují, jsou názorně vyobrazeny na diagramech v příloze „Diagramy Frameworku Mate".

7 Analýza a návrh systému

Jak bylo popsáno v kapitole 5.1 (Fáze životního cyklu), analýza a návrh jsou nedílnou součástí správného přístupu před implementací systému. Tato kapitola obsahuje požadavky na systém, dále neformální specifikaci a analýzu požadavků. Po té si uvedeme důležité modely prezentující analyzované požadavky.

Pro návrh diagramů byl použit výborný software Visual Paradigm for UML 7.0 Community Edition (VP). Čímž sice byly znemožněny některé pokročilé funkce tohoto IDE, nicméně pro návrh UML diagramů i přes omezení je tento software plně dostačující. Mimo VP je využito vizualizačního nástroje Visio z balíku Office 2007 společnosti Microsoft.

Zdrojová data diagramů jsou přístupna na přiloženém CD.

7.1 Neformální specifikace

Aplikace, která bude umožňovat správu projektů, možnost vytváření typizovaných testů a jejich nastavení, přiřazení osob k testům a dále zobrazení výsledků testů. Uživatelské prostředí by mělo být přívětivé

7.2 Analýza požadavků

Požadavky na systém rozdělíme do dvou částí, a sice na funkční a technické. V části první se budeme zabývat funkcionalitami systému ve smyslu: „co by aplikace měla umět“ a ve druhé části si popíšeme technické aspekty. Tedy to, na jakých operačních systémech, hardwaru a v jakém prostředí by systém měl fungovat.

7.2.1 Funkční požadavky

Aplikaci mohou ovládat tři typy uživatelů.

Administrátor, uživatel s nejvyššími privilegii má možnost přidávat a odebírat osoby a jejich účty, nebo jen blokovat přístup do systému.

Dále manager se stará o správu projektů. Kromě klasických voleb pro vytváření, úpravu a mazání záznamů, mu bude umožněno přiřadit k jednotlivým projektům typizované testy a k těmto testům dále možnost přiřazovat zvolené osoby.

Nakonec pracovník, kterému se po přihlášení do systému zobrazí přiřazené testy a dále bude mít možnost k jednotlivým testům přiřadit tzv. „subtesty“. Tyto subtesty představují jednotlivé výsledky měření zvolené charakteristiky.

Manager si zobrazí výsledky subtestů jako graf specifikovaného testu. Pokud bude chtít, může si graf přetáhnout například do Wordu stylem „drag and drop“.

Pro úpravu detailů projektů, testů a osob bude zpřístupněn rozšířený způsob popisu, například pomocí textového editoru, který umožňuje stylovat text (podtržení, velikost písma apod.).

7.2.2 Technické požadavky

Systém bude spustitelný v prostředí současných operačních systémů (Windows 2000/XP/Vista, různé distribuce Linuxu, případně Mac OS X). Není kladen nutný požadavek na ovládání aplikace přes internetový prohlížeč, proto bude vytvořena desktopová aplikace v prostředí Flex/AIR, kde by se

většina zdrojového kódu dala později využít pro vytvoření webové aplikace Flex/Web Application. Jako datové úložiště bude využit systém MySQL, který bude dostupný odkudkoliv.

Spojení s databází bude umožněno díky Amfphp, což je služba, která běží na PHP serveru. Díky ní je možné vzdáleně zasílat dotazy z AIR aplikace na Amfphp, která je předá dál do MySQL serveru. Vrácený objekt je dle potřeby transformován a pošle zpět aplikaci RemoteObject.

Aplikace by měla běžet na standardně vybaveném PC (případně platformě pro chod Mac OS X) a neměly by být kladeny žádné další nároky, mimo výše zmíněné, pro nutný provoz aplikace.

7.3 Glosář pojmů

Pro ujasnění pojmů použitých v návrzích UML je nutné definovat termíny a výrazy, se kterými se bude v následujícím text pracovat. Je tak vhodné učinit nejen kvůli nejednoznačnosti některých slov a frází, ale také kvůli ozřejmění smyslu a významu pojmů, se kterými se zadavatel projektu v běžném životě ani nesetká.

Pojmy s jejich popisy nalezneme v následující tabulce (Tabulka 7-1: Glosář pojmů):

Pojem	Popis
Aktér (Role)	Někdo nebo něco (může být například osoba, zařízení, další systém) mimo systém, který vykonává akci v systému.
Use-case	Akce v systému, kterou aktér může vykonat. Jeden případ použití.
Systém	V našem případě se jedná o aplikaci, ve které bude možné spravovat osoby, projekty, testy,…
Administrátor (Administrator)	Pověřená osoba pro správu osob, úpravu přihlašovacích údajů, práv osob a kontrolu logů.
Manažer (Manager)	Osoba, která se stará zejména o správu projektů a vytváření výsledků na základě zpracovaných testů přiřazených k projektům a k osobám.
Pracovník (Worker)	Osoba vyplňující testy, které jí jsou přiřazeny.
Osoba (Person)	Obecné označení role v systému.
Login	Neboli uživatelské jméno unikátní v daném systému. Též označení pro přihlášení do systému na základě loginu a hesla.
Heslo	Textový řetězec, který je ve spojení s loginem identifikujícím prvkem při přihlášení do systému.
Účet	Součást detailu osoby. Označuje trojici: práva, login a heslo. Práva vyplývají z typu role (administrátor, manažer, pracovník) a určují možnost vykonávat určité operace na základě příslušnosti k dané roli.
Detail	Součást popisu osoby, projektu či testu. Jde zejména o název, podnázev, popis, který může být proveden formátovacími nástroji (např. tučný text, změna velikost písma či barvy, apod.) a další nastavení, která jsou specifická pro daný typ modulu.
Projekt (Project)	Zastřešující modul pro skupinu testů spjatých k danému projektu.
Test	Hodnocení zkoumané metriky.
Výsledek (Result)	Uzavření projektu se slovním popisem a vyhodnocením.
Log	Záznam v databázi po provedené specifikované akce. Například přihlášení do systému.
Databáze (DB, Database)	Datové úložiště, v našem případě relační databáze. Zpravidla vystupuje jako „pasivní aktér“, myslíme tím, že většina akcí je neproveditelných bez komunikace s databází a bez získání dat v ní uložených.

Tabulka 7-1: Glosář pojmů

7.4 Diagramy UML

UML (Unified Modeling Language) je standardní modelovací jazyk pro tvorbu modelů při vytváření systému. Využijeme jeho možností při „grafické“ specifikaci modelů případů použití, návrhu databázového schématu (tzv. ER, entity-relationship, diagramu), konceptuálního diagramu tříd a jiných modelů.

Zákazníkovi (zadavateli projektu) je z následujících modelů nejprospěšnější diagram případů použití (také „use-case diagra“). Jeho význam a reálný příklad našeho systému nalezneme v následující podkapitole.

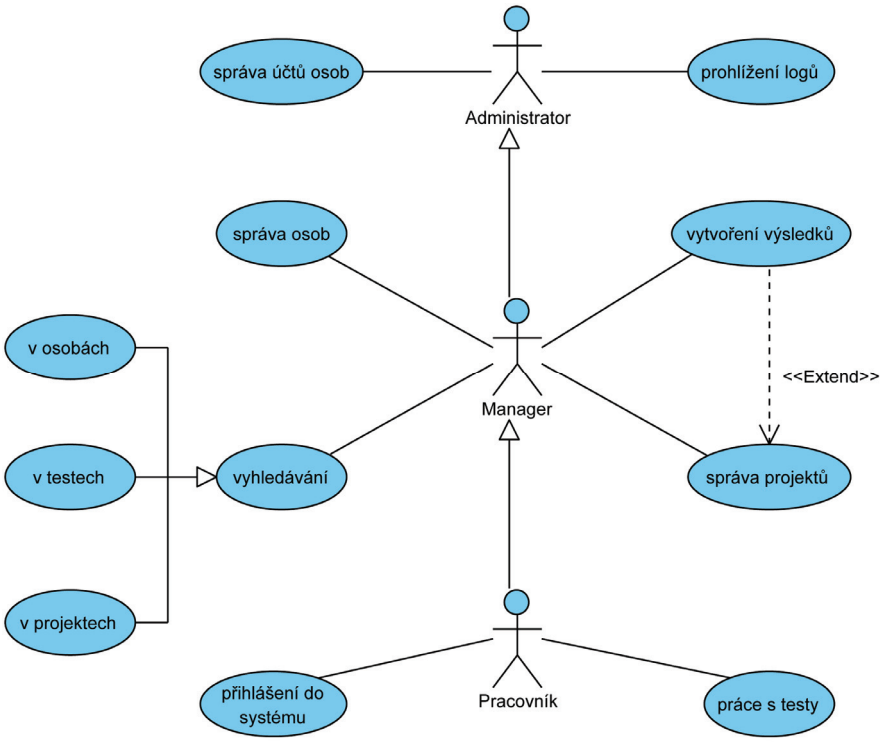
7.4.1 Diagramy případů použití

Pohled na celý systém

Na následujícím obrázku [Obrázek 7-1] je znázorněn celkový pohled na systém pomocí use-case diagramu. Jsou zde hlavní případy použití a všichni aktéři, jenž v systému mohou vystupovat.

Jde o následující případy použití:

- *Přihlášení a odhlášení ze systému*
- *Správa osob a správa účtů osob*
- *Správa projektů*
- *Práce s testy*
- *Práce s výsledky*
- *Vyhledávání*
- *Prohlížení logů*



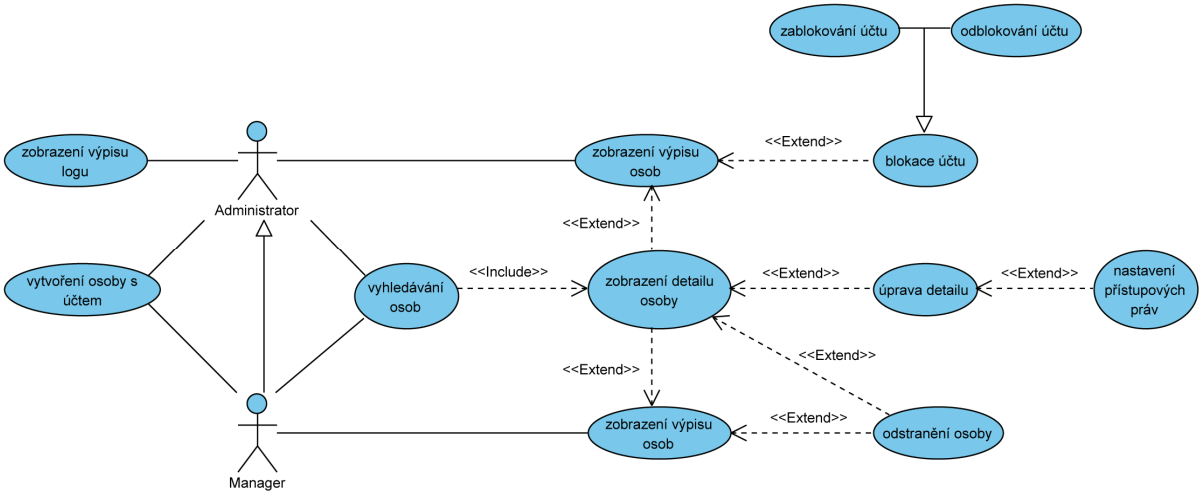
Obrázek 7-1: Diagram případů použití UC01 – Pohled na celý systém, hlavní diagram případů použití

Správa osob z pohledu administrátora a manažera

Popis diagramu

Na diagramu je znázorněna práce se záznamy osob. Tyto záznamy je možné spravovat buď administrátorem nebo manažerem. Ze strany manažera jde v podstatě o přidávání, úpravu a odstranění konkrétní osoby, případně celkový jejich výpis. Dále je možné měnit osobám login a heslo.

Administrátor navíc může účty blokovat.



Obrázek 7-2: Diagram případů použití UC02 – Správa osob z pohledu administrátora a manažera

Případ použití: UC02.01

Název:	Přihlášení do systému
ID:	UC02.01
Stručný popis:	Aktér se chce po spuštění aplikace přihlásit do systému. Aplikace mu tuto akci umožní na základě platných přihlašovacích údajů a také za předpokladu, že daný účet není blokován.
Hlavní aktéři:	Administrátor, manažer, pracovník
Vedlejší aktéři:	–
Předpoklady:	Aplikace se úspěšně připojila ke skriptu pod Amfphp a dále se úspěšně připojila k databázi.
Hlavní tok:	1. Případ použití se spustí, jakmile uživatel spustí aplikaci a je vyzván k zadání loginu a hesla. 2. Dokud je kombinace zadaného loginu a hesla neplatná 2.1 Zobrazení formuláře pro zadání loginu a hesla. 2.2 Ověření vyplnění nutných položek pro přihlášení do systému. 2.3 Ověření správnosti zadaných údajů s údaji z databáze. 3. Aplikace je zpřístupněna aktérovi, aktér dále může pracovat s aplikací na základě jeho privilegií.

Následné podmínky:	Aktér je přihlášen, aplikace umožní vykonávat jen takové operace, které odpovídají privilegiím právě přihlášeného aktéra.
Alternativní toky:	Login nebo heslo nejsou zadány. Login a heslo neodpovídá záznamu v databázi. Ukončení aplikace.

Specifikace vyjímek: UC02.02

Login nebo heslo nejsou zadány – Aplikace upozorní aktéra na tuto skutečnost informativním textem a dále zvýrazní ty položky, které je nutné vyplnit. Informativní text bude i u nutných položek.
Login a heslo neodpovídá záznamu v databázi – Aplikace opět upozorní aktéra, a vyzve k opětovnému zadání nových přihlašovacích údajů. Text položky s loginem zůstane zachován, heslo je vymazáno.
Ukončení aplikace – Dojde k ukončení aplikace.

Případ použití: UC02.03

Název:	Změna údajů u záznamu osoby
ID:	UC02.03
Stručný popis:	Aktér chce změnit údaje vybrané osoby.
Hlavní aktéři:	Administrátor, manažer
Vedlejší aktéři:	–
Předpoklady:	Aplikace umožní tuto akci je aktérovi, který je přihlášen do systému a má patřičná privilegia.
Hlavní tok:	1. Případ použití se spustí, jakmile uživatel zadá úpravu záznamu u vybrané osoby. 2. Dokud je validace zadaných údajů neúspěšná 2.1 Zobrazení formuláře pro zadání údajů o osobě. 2.2 Ověření správně zadaného e-mailu. 2.3 Ověření „neduplicity“ zvoleného loginu. 2.4 Ověření správně zadaných hesel (prvotního a jeho přepisu). 3. Aplikace uloží data do databáze na pozici upravované osoby. Aktér dál zůstává přihlášen v systému.
Následné podmínky:	Osoba je v databázi upravena na základě údajů vložených aktérem.
Alternativní toky:	Neplatný e-mail. Login je duplicitní (řetězec stejného obsahu již v databázi existuje). Heslo a jeho přepis nesouhlasí. Ukončení aplikace. Uložení údajů a zavření aktuálního okna/záložky.

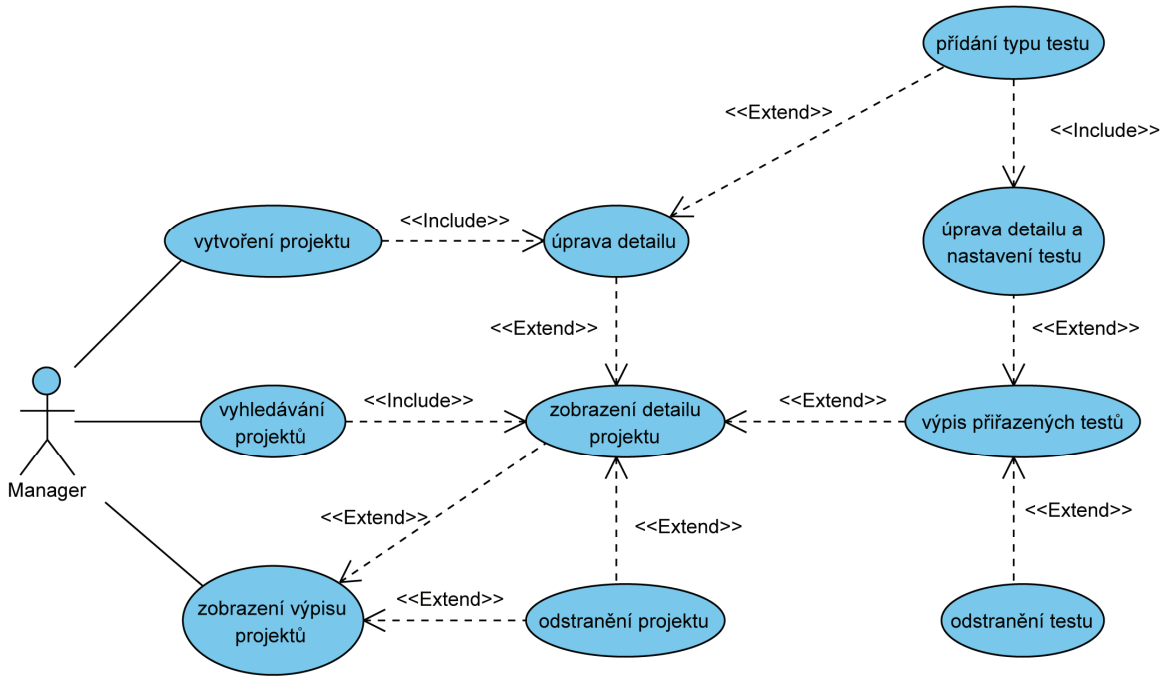
Specifikace vyjímek: UC02.04

Neplatný e-mail – E-mail musí odpovídat klasickému standardu, tedy tvaru: [a-zA-Z0-9\-_\.]*@[a-zA-Z0-9\-_\.]*\.[a-zA-Z0-9\-_\.]*.
Login je duplicitní (řetězec stejného obsahu již v databázi existuje) – Aplikace sama získává aktuální seznam všech loginů, které jsou uloženy v databázi v okamžiku změny loginu aktérem je porovnává s údaji v seznamu. Za předpokladu zvolení již existujícího loginu je na tuto skutečnost aktér upozorněn.
Heslo a jeho přepis nesouhlasí – Obě zadaná hesla musí souhlasit. Zadané heslo je zobrazeno skrytě kvůli bezpečnosti.
Ukončení aplikace – Dojde k ukončení aplikace, neuložené změny se tím pádem ztrácí.
Uložení údajů a zavření aktuálního okna/záložky – Aktér zvolí uložení údajů a zároveň zavření aktuálního okna/záložky.

Správa projektů z pohledu manažera

Popis diagramu

Manažer je pověřenou osobou pro práci s projekty – tzn. jejich vytváření, nastavení testů apod.
Pokud aktér zvolí vytvoření projektu, má možnost projekt popsat a po uložení i přiřadit typizovaný test. Manažer dále upravuje nastavení testu a nakonec jej uloží. Projekty si může nechat vypsat případně vyhledat, po té přejde na detail projektu, kde se ihned zobrazí přiřazené testy.



Obrázek 7-3: Diagram případů použití UC03 – Správa projektů z pohledu manažera

Případ použití: UC03.01

Název:	Přiřazení testu zvolenému projektu
ID:	UC03.01
Stručný popis:	Aktér chce přiřadit test projektu, který je již vytvořen.

Hlavní aktéři:	Manažer
Vedlejší aktéři:	–
Předpoklady:	Aktér je přihlášen, zvolený projekt je uložen v databázi (v aplikaci lze poznat tak, že přiřazené ID > 0).
Hlavní tok:	1. Případ použití se spustí, pokud manažer zvolí úpravu detailu projektu. 2. Manažerovi je ihned zobrazen seznam přiřazených testů a zároveň je mu umožněno přiřadit typizovaný test. 3. Manažer vybere test. 4. Ten je ihned uložen do databáze a přiřazen projektu. 5. Manažer upravuje popis a nastavení testu. 6. Uloží test.
Následné podmínky:	Aktér je přihlášen. Data související s testem uložena do databáze, pohledy zobrazující přiřazené testy jsou aktualizovány.
Alternativní toky:	Uzavření okna/záložky testu bez uložení.

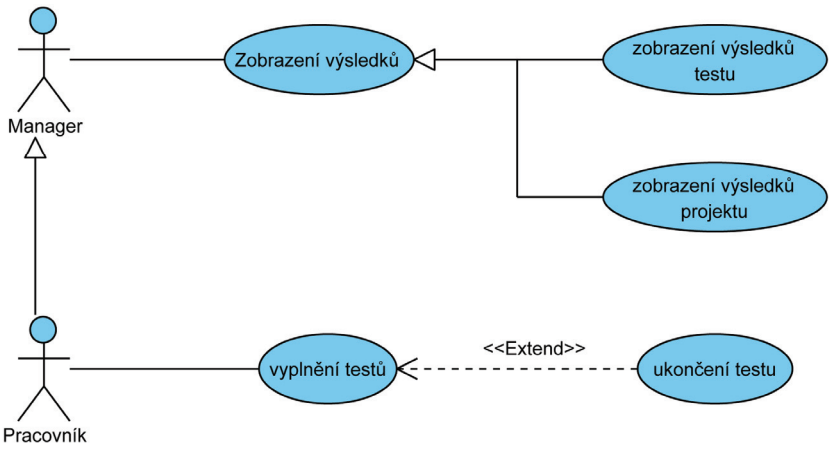
Specifikace vyjímek: UC03.02

Uzavření okna/záložky testu bez uložení – Dojde ke ztrátě neuložených dat, zůstávají platná ta, která byla uložena naposledy.

Práce s testy a s výsledky

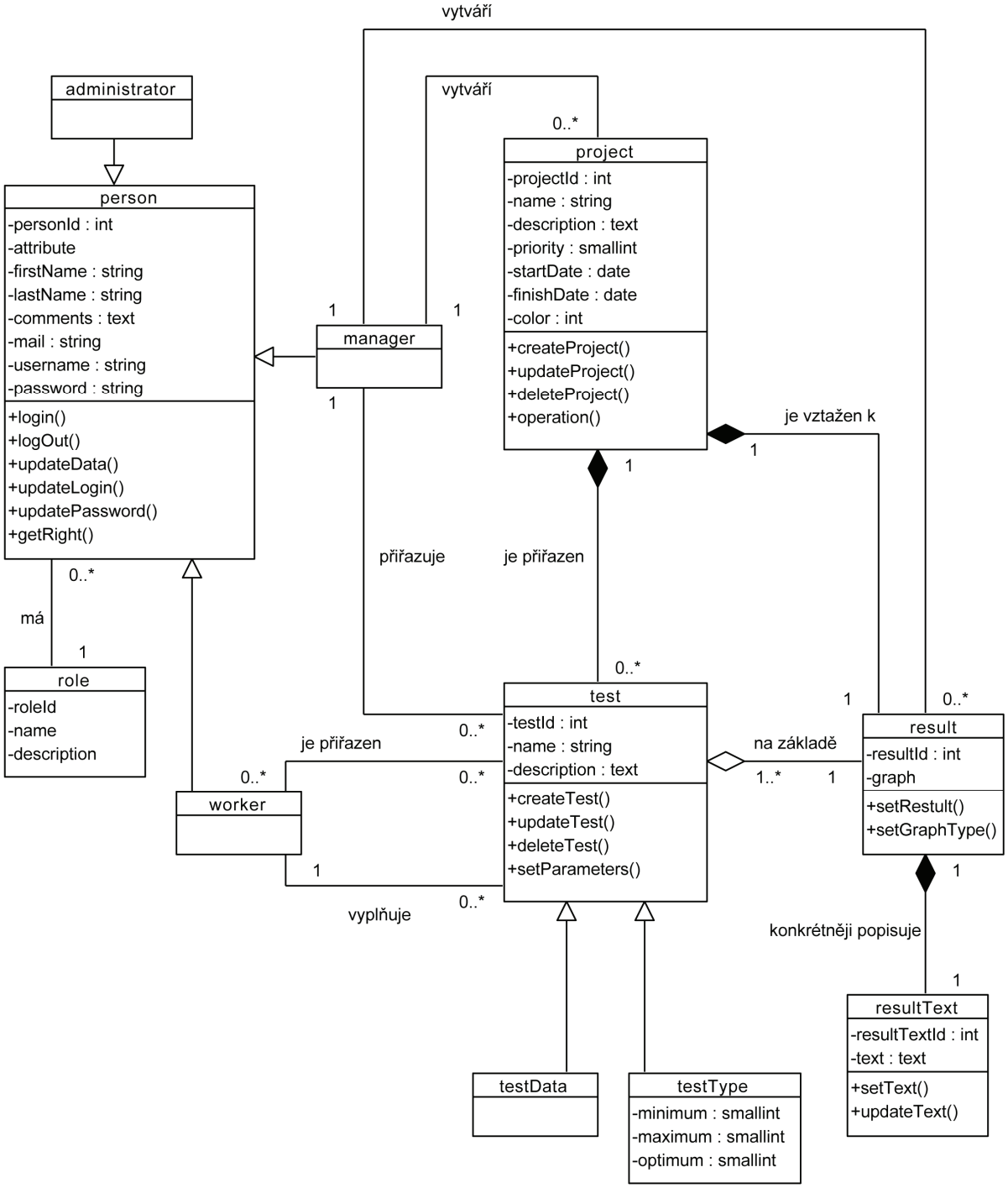
Popis diagramu

Pracovník má povinnost vyplnit testy přiřazené k daným projektům, může je nakonec ukončit. Manažer tedy ví, nakolik výsledky zobrazené k danému projektu odpovídají zlomku vyplněných testů.



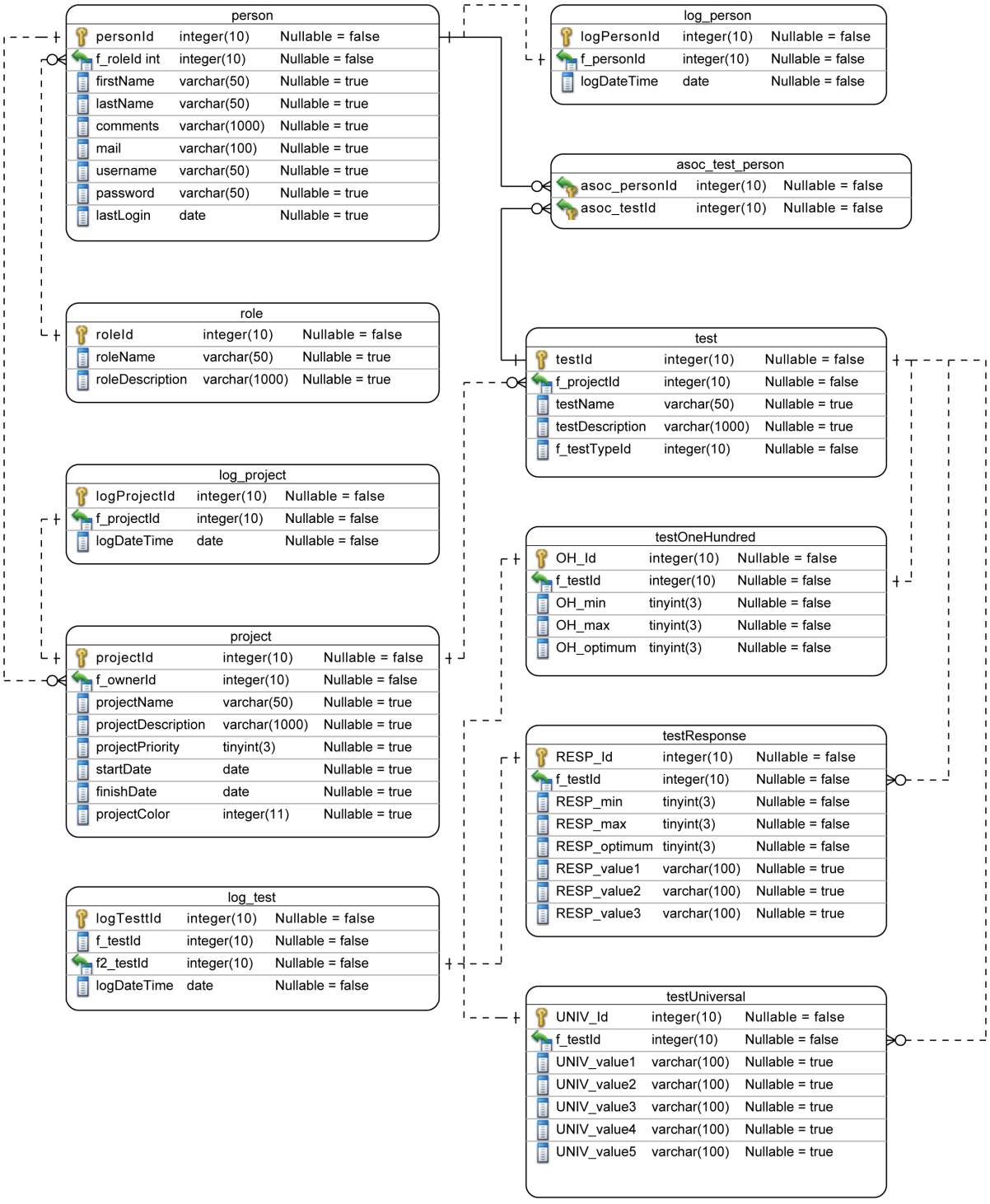
Obrázek 7-4: Diagram případů použití UC04 – Práce s testy a s výsledky

7.4.2 Konceptuální diagram tříd



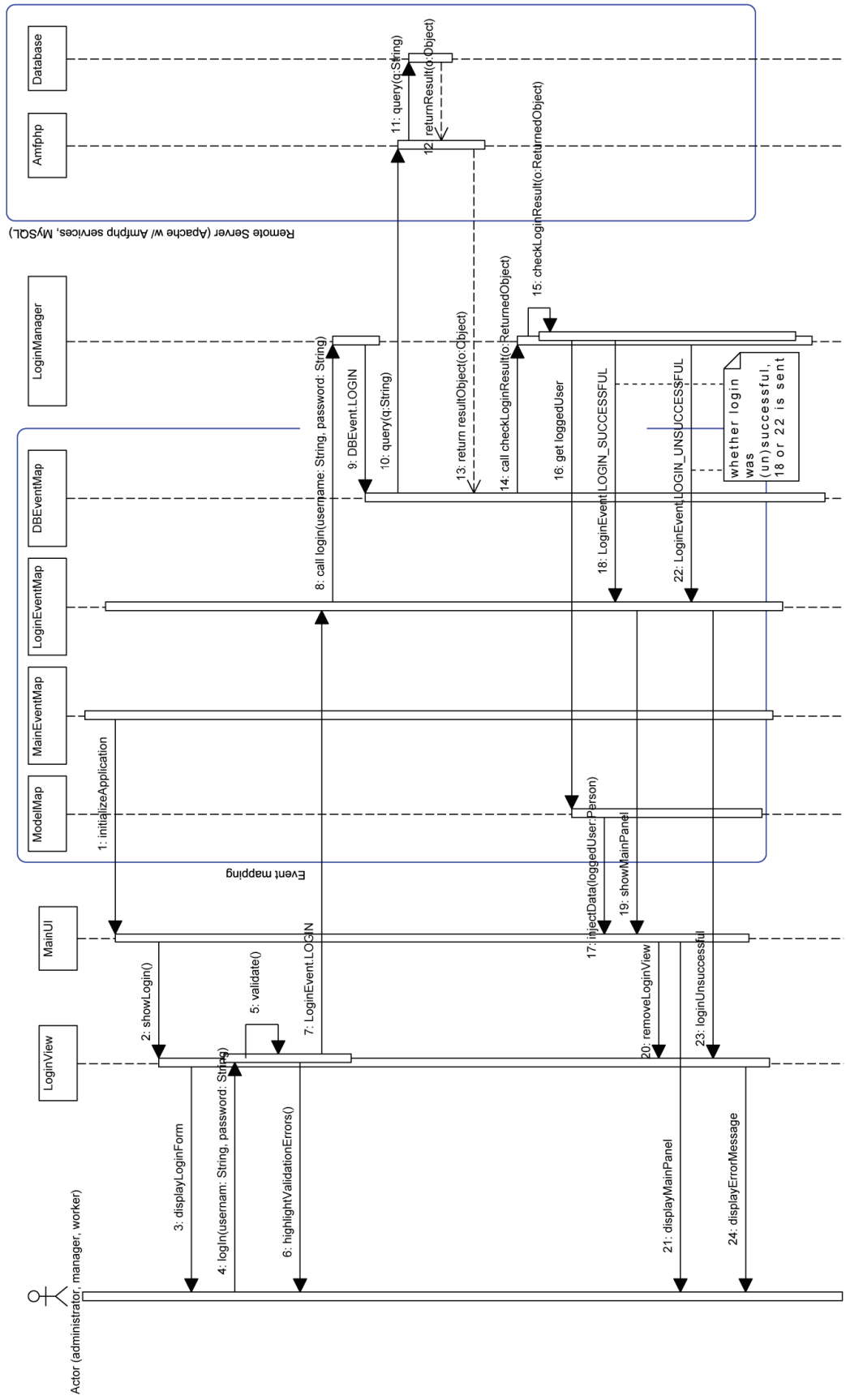
Obrázek 7-5: Konceptuální diagram tříd systému

7.4.3 Návrh databázového schématu



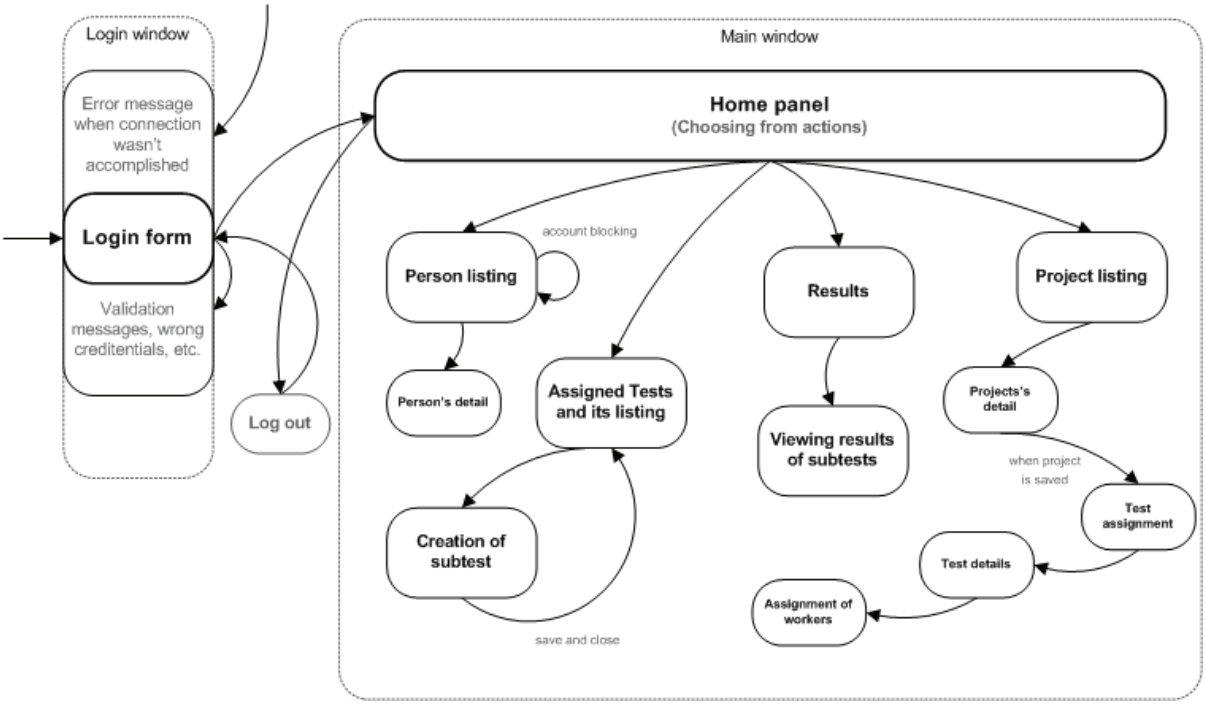
Obrázek 7-6: Návrh databázového schématu

7.4.4 Sekvenční diagramy



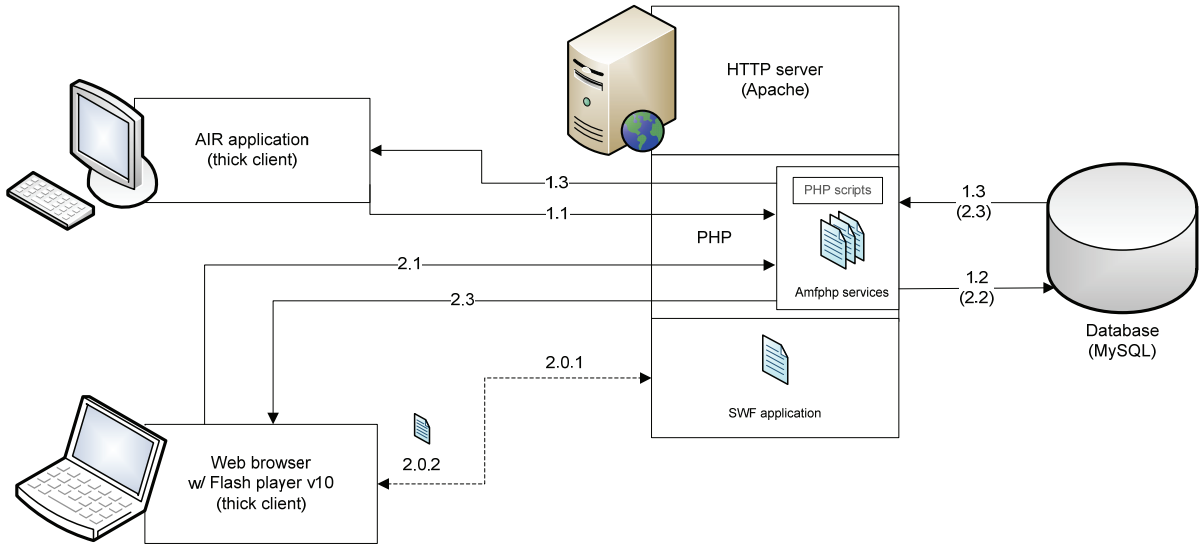
Obrázek 7-7: Sekvenční diagram: přihlášení uživatel

7.4.5 Diagram návaznosti obrazovek



Obrázek 7-8: Diagram návaznosti obrazovek

7.4.6 Návrh architektury aplikace



Obrázek 7-9: Návrh architektury aplikace

8 Implementace

Tato kapitola je rozdělena na tři podkapitoly, tu základní, která odpovídá popisu implementace aplikace a závěrečných dvou – instalace a testování. První kapitola reflektuje balíčkové rozdělení navržené aplikace s využitím frameworku Mate a návrhového vzoru „Inversion of Control“.

8.1 Balíčky

Mimo balíčky je vhodné se zmínit o realizaci GUI aplikace. Ta využívá skinu aplikace Salesbuilder[19] a některých dalších funkcí, které jsou vhodně použity v uvedeném řešení.

Pro lepší pochopení hierarchie a filozofii balíčků, autor doporučuje nastudování diagramů Mate z přílohy (Obrázek 9-9 a Obrázek 9-10)[20]. Vhodným zdrojem je i diagram sekvence (Obrázek 7-7).

8.1.1 map

Ústřední „mapou“ je třída `EventMap`. Monitoruje všechny vyslané události a zároveň události vysílá, takže je centrálním bodem kooperace mezi jinými třídami balíčků. Její MXML kód se skládá z různých konstrukcí, jako například:

```
<EventHandlers type="{PersonEvent.DELETE}">
        <MethodInvoker generator="{PersonManager}" method="deletePerson"
        arguments="{[event]}"/>
</EventHandlers>
```

První řádek se zpracuje na základě vyslané události a volá metodu `deletePerson` ve třídě `PersonManager`, jako argument zašle parametry přijaté události. Dále se zde mimo `MethodInvoker` objevují `EventAnnouncer`:

```
<EventAnnouncer generator="{NavigationEvent}" type="{NavigationEvent.EXIT_HANDLER}"/>
```

Tyto tagy vysílají novou událost. V tomto případě se jedná o ukončení aplikace.

Mimo `EventMap` se v balíčku ještě vyskytují `LoginEventMap`, `ModelMap` a `DBEventMap`. Poslední dva zmíněné jsou pro chod aplikace poměrně specifické. `ModelMap` vykonává tzv „injecting“ (plnění dat apod.). Typickým příkladem je blok:

```
<Injectors target="{PersonDetail}">
        <PropertyInjector targetKey="rolesList" source="{DBManager}"
        sourceKey="rolesList"/>
        <PropertyInjector targetKey="usernamesList" source="{PersonManager}"
        sourceKey="usernamesList"/>
 </Injectors>
```

Znamená, že „injektovat“ se má třída s názvem `PersonDetail` balíčku `view` daty `rolesList` a `usernamesList` pomocí tříd `DBManager` respektive `PersonManager`, pokud se `rolesList` nebo `usernamesList` změní. V praxi to znamená, že při úpravě detailu osoby jsou neustále aktuální data o uživatelských jménech a typech práv.

`DBEventMap` se stará o komunikaci s Amfphp – zasílá dotazy, které třída Amfphp dále předá do MySQL a Amfphp zpět posílá objekt s výsledkem dotazu. Základním `RemoteObject` je:

[19] http://coenraets.org/blog/2009/04/new-version-of-salesbuilder-with-ribbit-integration-source-code-available/
[20] Pro ještě hlubší pochopení je také vhodné zhlédnout další dva diagramy: Obrázek 9-11 a Obrázek 9-12.

```
<mx:RemoteObject
        id="dbQuery"
        source="dip.DB"
        destination="amfphp"
        showBusyCursor="true"/>
```

A přihlášení pak vypadá následovně:

```
<EventHandlers type="{DBEvent.LOGIN}">
        <RemoteObjectInvoker instance="{dbQuery}" method="query"
        arguments="{event.query}">
                <resultHandlers>
                        <MethodInvoker generator="{LoginManager}"
                        method="checkLoginResult" arguments="{resultObject}"/>
                </resultHandlers>
        </RemoteObjectInvoker>
</EventHandlers>
```

Opět se naslouchá na událost `DBEvent.LOGIN` a po té zavolá instance `RemoteObject` a předá se mu parametr vyjádřen SQL dotazem. Výsledek se předá třídě `LoginManager` a metodu `checkLoginResult` s argumentem `resultObject`. Podrobnější informace naleznete v nápovědě pro Mate[21].

8.1.2 view

Balíček obsahující všechny UI prvky. Jsou zde definovány panely projektů, testů a osob. Výpisové tabulky, validátory, grafy a mnoho jiného. Základem je MXML kód, který je však často doplněn o funkce, které obstarávají události vyvolané při různých akcích vykonaných ve `view` třídách. Některé konstrukce použité například ve `ProjectTestListing`:

```
…
        public var oID:String = SHA1.hash(Math.random().toString());
        [Bindable]
        public var assignedTests:ArrayCollection = new ArrayCollection;

        public function creationCompleteHandler():void {
                getAssignedTests();
        }
…
```

Proměnná `oID` obsahuje náhodně vygenerovanou hodnotu, která se používá při zpětném injektování konkrétního `view` jako unikátní `oID`. `assignedTests` je kolekce, který má schopnost být injektován (viz popis `ModelMap` v předchozí kapitole). `creationCompleteHandler` je spuštěno při ukončení vytvoření `view` a zavolá funkci pro získání přiřazených testů. Pokud jsou testy injektovány, „listener“ přijme data na základě `oID` a vykoná na získaný objekt cyklus:

```
…
for(var i:int = 0; i < event.data.length; i++) {
  assignedTests.addItem(TestADG.mapObjectAssigned(event.data.getItemAt(i)));
}
…
```

Projde všechny pozice objektu a transformovaná data přidá do kolekce `assignedTests`. Transformace probíhá následovně: vybrání položky z objektu (`event.data.getItemAt(i)`),

[21] http://mate.asfusion.com/page/documentation/tags/eventmap

namapování pomocí statické metody `mapObjectAssigned` třídy `TestADG` na objekt typu `TestADG` (model vhodný pro navržený `AdvancedDataGrid`) a nakonec přidá do kolekce.

Jako ukázka by předchozí příklad stačil, následující balíček je `events`.

8.1.3 events

Všechny třídy tohoto balíčku jsou události rozšiřující `Event` z `flash.events.Event`. Opět je zde vidět nezávislost na frameworku Mate. Třídy obsahují kódy konstrukcí podobné následující z DBEvent:

```
…
public static const ERROR_CONNECTION:String = "errorConnection";

        public var name:String;
        public var query:String;
        public var oID:String;
        public var returnSubtest:int = 0;

        public function DBEvent(type:String, bubbles:Boolean=true,
        cancelable:Boolean=false)
        {
                super(type, bubbles, cancelable);
        }
…
```

Jde tedy o definici uživatelských událostí podobných těm nativním `flash.events.Event`.

8.1.4 manager

Třídy s koncovkou `*manager` jsou zřejmě tím nejdůležitějším funkcionálním pojítkem mezi daty a `DBEventMap`. Každá třída reflektuje logický model jako osoba, projekt, data apod. a obsahuje obsáhlou sadu metod pro práci se získanými objekty s databáze a dále pro konstrukci dotazů pro CRUD[22] s databází.

Jako příklad si uvedeme metodu `getProjectTests` třídy `TestManager`, což je metoda pro zaslání události s parametrem obsahujícím dotaz k získání všech přiřazených testů podle ID projektu.

```
public function getProjectTests(event:TestEvent):void {
         var db:DBEvent = new DBEvent(DBEvent.GET_TESTS);
         db.query =     "SELECT * FROM test " +
                        " RIGHT JOIN project ON project.projectId = test.f_projectId " +
                        " WHERE test.f_projectId = " + event.projectId + " " +
                        " AND test.isSubtest = 0";
         db.oID = event.oID;
         dispatcher.dispatchEvent(db);
}
```

Nakonec se událost odešle. Opět jde o třídy na Mate nezávislé, nevolá se žádná konkrétní metoda a ani funkce Mate nebo něco z `EventMap`.

8.1.5 model

Balíček `model` obsahuje třídy, jež jsou datovými modely reálných modelů. Opět to mohou být osoba, projekt, test atd. Jsou zde parametry typu jméno, příjmení, heslo a jiné. Dále modely obsahují statické metody pro mapování objektů na model:

[22] Create, read, update, delete – základní operace nad perzistentními daty.

```
public static function mapObjectToPerson(obj:Object):Person {
        var p:Person = new Person;

        p.personId     = obj.personId;
        p.firstName    = obj.firstName;
        p.lastName     = obj.lastName;
        p.mail         = obj.mail;
        p.comments     = obj.comments;
        p.username     = obj.username;
        p.password     = obj.password;
        p.roleId       = obj.roleId;
        p.roleName     = obj.roleName;
        obj.personBlocked == 1 ? p.blocked = true : p.blocked = false;

        return p;
}
```

8.1.6 components a util

Poslední dva balíčky obsahují různé doplňující metody, například upravené TextInputy, centrovatelné CheckBoxy, definice validátorů a mnoho jiného.

8.2 Instalace

Instalaci je nutné brát z více úhlů pohledu. Buď jako samotnou instalaci klienta, čemuž odpovídají kapitoly 8.2.1 a 8.2.2 nebo jako kompletní instalaci, kdy je nutné mít zprovozněn web server (Apache, ISS nebo jiný) s PHP a MySQL. Pak je nutné ještě nahrát službu Amfphp (8.2.3). Instalace webového serveru je nad rámec textu této práce a je předpoklad, že tuto věc uživatel zvládne, pokud chce provozovat databázový server.

8.2.1 AIR

Aplikace je psána tak, aby běžela v prostředí Flex. Abychom mohli spustit instalaci, ve většině případů stačí spustit samoinstalační soubor `dip_app.air`, který se nachází ve složce [`/app`] na přiloženém CD. Pokud se tak nestane, je nutné si prvně stáhnout a nainstalovat Adobe AIR[23]. Po tomto kroku by instalace měla proběhnout bez problému. Tyto jednoduché kroky jsou pro Windows XP.

Pod operačním systémem Linux, resp. distribuci Ubuntu, je postup následující:

1. Stáhnout Adobe AIR installer ze stránek Adobe
2. U staženého instalátoru odstranit koncovku (`AdobeAIRInstaller.bin -> AdobeAIRInstaller`)
3. Spustit instalátor
4. Pomocí AdobeAIRInstaller (nainstalovaného) spustit instalaci `dip_app.air`.

U distribuce Gentoo se jednoduchým způsobem instalace nezdařila, nicméně existují způsoby, jak AIR aplikace pod Gentoo spustit[24].

Odinstalace je například u Windows XP proveditelná odstraněním nainstalované aplikace z „`Ovládací panel->Přidat nebo odstranit programy`“ (v anglické verzi „`Control Panel->Add or Remove Programs`“).

[23] Na adrese http://get.adobe.com/air/, zde si také můžete vybrat instalaci pro operační systémy Windows, Linux a Mac OS X.

[24] http://forums.adobe.com/message/23483#23483

8.2.2 SWF

Aplikaci SWF je možné spustit z běžného webového prohlížeče s pluginem Flash Player minimálně verze 9 (někdy byla potřeba verze 10, záleží na nastavení OS a prohlížeče). SWF aplikaci půjde s největší pravděpodobností spustit přes `[/app/index.html]`.

Aplikace bude také přístupna online na privátní adrese http://dip.soulwasted.net (toto je i server, kde běží PHP, Amfphp a MySQL).

8.2.3 Amfphp

Pro chod Amfphp stačí do rootu na PHP server nahrát Amfphp z adresáře `[/third_party]`. Dále nahrát složku s PHP skripty z adresáře `[/app/amfphp]` na PHP server do adresáře `[/amfphp/services]`.

8.2.4 Nápověda

Základní poznatek pro uživatel ještě před přečtením manuálu, by mělo být to, že pokud je místo kurzoru zobrazen „budík“, provádí se komunikace s databází a tímto způsobem je tato informace uživateli dána najevo.

Jednoduchý uživatelský manuál je uložen ve formě HTML v adresáři [`/app/help]`.

8.3 Testování

Testování probíhalo po celou dobu pod operačním systémem Windows XP SP3 EN. Po dobu vývoje aplikace se neprojevily žádné negativní známky technologie AIR, čímž se potvrdila jeho stabilnost.

Další systém, na kterém testování proběhlo, bylo Ubuntu verze 8.10 s GNOME Desktop 2.24.1, které běželo pod VMware Workstation 6.5.1 na Windows XP SP3. I zde aplikace běžela stejně dobře jako u prvně zmíněného testovacího prostředí.

Co se týče spouštění SWF aplikace, tak se neprojevily problémy v žádném z webových prohlížečů (Firefox 3.X, Internet Explorer 7 a 8, Opera 9.62, Safari 4b – WinXP SP3), jen je potřeba mít správnou verzi Flash Playeru.

Server, kde běží PHP a MySQL, má tyto parametry: Linux ego 2.6.18-6-xen-amd64, Apache 2.0, PHP 5.2.0-8, MySQL 5.0.51a a ještě Amfphp verze 1.9.beta.20080120.

Tímto jsme zakončili poslední kapitolu – Implementace – a následuje závěrečné zhodnocení.

9 Závěr

Cílem diplomové práce bylo seznámit se s problematikou systémů řízení jakosti. Dalším bodem bylo zhodnocení současných možností hodnocení kvality při návrhu a tvorbě softwarových produktů, zejména nástrojů a prostředků pro vývoj webových aplikací. Problematika managementu jakosti je rozsáhlým tématem a pro nalezení konkrétních požadavků je potřeba důkladnější analýzy v dané oblasti.

Management jakosti není pojmem pouze pro oblast řízení softwarových procesů. Zasahuje do všech oblastí, kde je nutné zajistit kvalitu produktu při všech stádiích jeho vzniku. Tedy od okamžiku prvotního kontaktu se zákazníkem, přes fáze příprav (výrobku, výroby, materiálu) až po realizaci, nasazení, servis atd. Řada nedostatků při vytváření produktů vyjde najevo až když známe výsledek určité operace, specifické kombinace kroků anebo všech realizačních procesů. Filozofie moderního managementu tkví právě v koordinaci a řízení procesů tak, aby byly kvalitní, což implikuje jako výsledek i kvalitní produkt. Tento přístup se dá aplikovat nejen na realizaci produktu, ale i na jeho vývoj, hodnocení a zlepšování. Vše pomocí procesů.

Praktická část práce je zaměřena na výběr vhodného nástroje pro vývoj webových aplikací s přihlédnutím k realizaci požadavků na programovou podporu softwarových procesů. Aplikace není navržena pro konkrétní systém řízení jakosti, je navržena univerzálně tak, aby splňovala základní podmínky pro práci s metrikami při správě a hodnocení softwarových procesů na principu využití některé z norem nebo modelů. A to buď těch, které v práci zmíněny byly, nebo i těch, které při hodnocení využívají různé metriky. Aplikace je plně funkční dle specifikace, navíc provedena ve dvou variantách. Pro prostředí AIR (desktopová verze) a jako SWF aplikace (pro webové prohlížeče). Vedlejším efektem při vypracování diplomové práce bylo velice dobré seznámení s vybranou technologií Flex, ve kterém byly oba typy vyvíjeny, a se kterým autor neměl dříve žádnou zkušenost.

Ačkoliv osnova práce plně neodpovídá bodům zadání diplomové práce, uspořádání i obsah kapitol na sebe logicky navazují a odráží předmět bodů zadání práce. Osnova byla také konzultována s vedoucí – Ing. Šárkou Květoňovou

Co se týče možnosti rozšíření vyvíjené aplikace, je zde mnoho prostoru pro změny. Ať už se jedná o konkretizaci funkcionalit podmíněných potřebám systému řízení jakosti, nebo o rozšíření možných typů nástrojů řízení jakosti, není tento požadavek problémem vzhledem k propracované modularitě aplikace. Toho bylo docíleno nejen použitím frameworku Mate, ale také systematickým přístupem při návrhu aplikace na základě získaných znalostí během studia.

Literatura

[1] KAN, Stephen H. *Metrics and Models in Software Quality Engineering*. 2nd edition. [s.l.] : Addison Wesley, 2002. ? s. ISBN 0-201-72915-6.

[2] CMMI Product Team. *CMMI® for Development, version 1.2 : Improving processes for better products*. [s.l.] : [s.n.], 2006. 561 s. Dostupný z WWW: <http://www.sei.cmu.edu/publications/documents/06.reports/06tr008.html>.

[3] KULPA, Margaret K., JOHNSON, Kent. A. *Interpreting the CMMI : A Process Improvement Approach*. 2nd edition. [s.l.] : [s.n.], 2008. 404 s. ISBN 978-1-4200-6052-2.

[4] MULAČ, Viktor. CMMI - Cesta ke zlepšení zralosti organizace IT při budování IS. *Role standardů a norem při zvyšování zralosti organizací IT* [online]. 2007 [cit. 2009-02-09]. Dostupný z WWW: <http://www.cssi.cz/cssi/system/files/all/sem-zralost-mulac.pdf>.

[5] MADACHY, Raymond J. *Software Process Dynamics*. [s.l.] : IEEE Press, 2008. 601 s. ISBN 978-0-471-27455-1.

[6] *Akademie : Kvalita - Systém kvality* [online]. 2006-2008 [cit. 2009-02-10]. Dostupný z WWW: <http://www.vlastnicesta.cz/akademie/kvalita-system-kvality/>.

[7] *ISO 9000 and ISO 14000* [online]. 2009 [cit. 2009-02-10]. Dostupný z WWW: <http://www.iso.org/iso/iso_catalogue/management_standards/iso_9000_iso_14000.htm >. Http://www.iso.org/.

[8] MACIASZEK, Leszek A., LIONG, Bruc Lee. *Practical Software Engineering : A Case Study Approach*. 1st edition. Harlow : Pearson Education Limited, 2005. 825 s. ISBN 0321204654.

[9] UČEŇ, Pavel. *Metriky v informatice : Jak objektivně zjistit přínosy informačního systému*. 1. vyd. Praha : Grada Publishing, 2001. 139 s. ISBN 80-247-0080-8.

[10] LAIRD, Linda M., BRENNAN, M. Carol. *Software Measurement and Estimation : A Practical Approach*. 1st edition. New Jersey : IEEE Computer Society, 2006. 257 s. ISBN 0-471-67622-5.

[11] HORÁLEK, Vratislav. *Jednoduché nástroje řízení jakosti I : Výstup z projektu podpory jakosti č. 5/16/2004*. Praha : Národní informační středisko pro podporu jakosti, 2004. 87 s. Dostupný z WWW: <http://www.npj.cz>. ISBN 80-02-01689-0.

[12] PLÁŠKOVÁ, Alena. *Jednoduché nástroje řízení jakosti II : Výstup z projektu podpory jakosti č. 5/16/2004*. Praha : Národní informační středisko pro podporu jakosti, 2004. 72 s. Dostupný z WWW: < http://www.npj.cz>. ISBN 80-02-01690-4.

[13] ZENDULKA, Jaroslav, et al. *Analýza a návrh informačních systémů AIS : Studijní opora*. [s.l.] : [s.n.], 2006. 178 s.

[14] HILLERSON, Tony. FrameworkQuest 2008 Part 5 : Mate, the Pure MXML Framework. *InsideRIA* [online]. 2008 [cit. 2008-05-21]. Dostupný z WWW: <http://www.insideria.com/2008/12/frameworkquest-2008-part-5-mat.html>.

[15] OFFUTT, Jeff. Quality Attributes of Web Software Applications. In *IEEE SOFTWARE*. [s.l.] : [s.n.], 2002. s. 25-32. doi:10.1109/52.991329.

[16] BOREK, Bernard. Adobe Flex - co je a co není. *Interval.cz : webdesign a e-komerce denně* [online]. 2008 [cit. 2009-05-21]. Dostupný z WWW: <http://interval.cz/clanky/adobe-flex-co-je-a-co-neni/>. ISSN 1212-8651.

[17] *Adobe® Flex® 3.3 Language Reference* [online]. 2009. 2009 , Thu Feb 26 2009, 03:20 PM -05:00 (index) [cit. 2009-05-20]. Dostupný z WWW: <http://livedocs.adobe.com/flex/3/langref/index.html>.

[18] *Adobe Flex 3 Help* [online]. [2009] [cit. 2009-05-20]. Dostupný z WWW: <http://livedocs.adobe.com/flex/3/html/index.html>.

[19] LOTT, Joey, PATTERSON, Danny. *Advanced ActionScript 3 with Design Patterns,*. 1st enl. edition. [s.l.] : Adobe Press, 2006. 304 s. ISBN 0-321-42656-8.

[20] COLE, Alaric. *Learning Flex 3 : Getting up to Speed with Rich Internet Applications*. 1st edition. [s.l.] : O\'Reilly, 2008. 303 s. ISBN 978-0-596-51732-8.

[21] AHERN, Dennis M., CLOUSE, Aaron, TURNER, Richard. *CMMI® Distilled : A Practical Introduction to Integrated Process Improvement*. 3rd edition. [s.l.] : Addison Wesley Professional, 2008. 288 s. ISBN 0-321-46108-8.

[22] VEBER, Jaromír, et al. *Řízení jakosti a ochrana spotřebitele*. 2. aktualiz. vyd. [s.l.] : Grada Publishing, 2006. 204 s. ISBN 978-80-247-1782-1.

Seznam příloh

CD

Přiložené CD obsahuje:

- *Informační soubor readme.txt*
- *Kompletní zdrojové kódy vytvořené aplikace*
- *Instalaci aplikace (dip_app.air)*
- *Runtime environment Adobe AIR pro Windows, Linux a Macintosh ve verzi 1.5.1 a Amfphp*
- *SQL skript pro instalaci databáze*
- *PHP skript pro Amfphp services*
- *Text této diplomové práce ve formátech `*.doc` a `*.pdf`*
- *UML diagramy ve formátu pro Visual Paradigm*
- *Ostatní diagramy pro Microsoft Visio*
- *Diagramy frameworku Mate, screenshoty aplikace*

Adresářová struktura:

```
[/]     {root}
        readme.txt          // informační soubor
[/app]                      // instalace dip_app.air a index.html
[/app/amfphp]               // PHP skripty pro Amfphp
[/app/sql]                  // SQL skript
[/app/source-air]           // projekt Flex Builderu se zdrojovými kódy
                            // pro AIR
[/app/source-swf]           // projekt Flex Builderu se zdrojovými kódy
                            // pro SWF
[/app_help]                 // uživatelská příručka
[/doc]                      // text této diplomové práce (doc a pdf)
[/doc/mate_diagrams]        // diagramy frameworku Mate
[/doc/other_diagrams]       // ostatní diagramy (pro Microsoft Visio)
[/doc/screenshots]          // screenshoty aplikace
[/doc/uml_diagrams]         // UML diagramy pro Visual Paradigm
[/third_party]              // Amfphp, AdobeAIRInstall, Adobe Acrobat
                            // Reader
```

Seznam obrázků

Seznam tabulek

Seznam použitých pojmů

.Net Framework Softwarový framework pro OS Windows. Obsahuje spouštěcí rozhraní a potřebné knihovny pro běh aplikací.

Actionscript Skriptovací jazyk, vychází z ECMAScript. Využívá se zejména pro vývoj

Capability Maturity Model Integration CMMI. Je model kvality organizace práce určený pro vývojové týmy. Model má 5 úrovní zralosti a prostřednictvím "auditu" se hodnotí na jaké z úrovní kvalita práce týmu je. Model CMMI je volně dostupný na Internetu [wikipedia.org].

Common Language Runtime „Běhový“ (runtime) systém .Net Frameworku

Databáze Datová základna) je určitá uspořádaná množina informací (dat) uložená na paměťovém médiu [wikipedia.org].

(Adobe) Flex Adobe Flex je sada technologií, vytvořená Adobe Systems pro vývoj multiplatformních RIA aplikací (Rich Internet Application) založených na Adobe Flash [wikipedia.org].

ECMAScript Skriptovací jazyk (specifikace v ISO/EIC 16262). Nejznámějšími dialekty jsou JavaScript a JScript.

Flash Adobe Flash, dříve Macromedia Flash, je multimediální platforma, kde „flashové“ aplikace zpočátku spadaly do kategorie animací, interaktivních reklam, poslední dobou, zejména díky ActionScript 3.0, se výsledné produkty řadí do skupiny RIA.

Framework Softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů.

Ishikawa diagram Diagram příčin a následků.

JavaFX Softwarová platforma pro tvorbu RIA, které mohou běžet na různých zařízeních.

Mate Framework pro Flex. Je pojmenován dle nápoje maté a rovněž se tak i čte.

Mysql Databázový multiplatformní systém, existuje ve dvou verzích: GPL (bezplatná licence) a Enterprise (komerční).

Oracle Moderní multiplatformní databázový systém s velice pokročilými možnostmi zpracování dat, vysokým výkonem a snadnou škálovatelností [wikipedia.org].

Plugin Zásuvný modul (z angl. „plug-in“), který rozšiřuje funkčnost aplikace.

Rich Internet Applications Webové aplikace, které mají charakteristiky desktopových aplikací.

Ruby on Rails Framework pro vývoj webových aplikací napojených na DB. Využívá MVC.

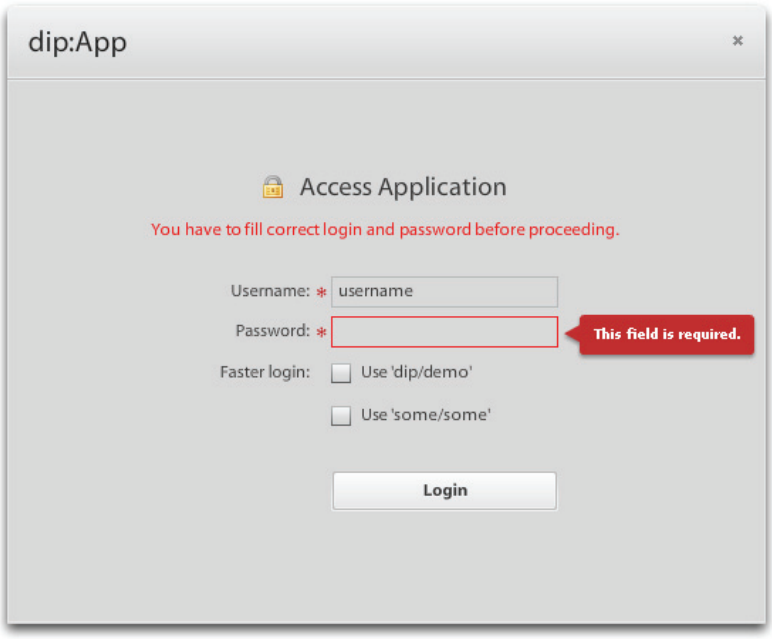
Scrum	Jedna z metod agilního vývoje.
Silverlight	Plug-in společnosti Microsoft pro internetové prohlížeče pro RIA. Silverlight je postaven na technologii Windows Presentation Foundation (WPF) a .NET Framework 3.0 [wikipedia.org].
Tenký klient	Zpravidla zařízení s internetovým prohlížečem, který je využit jako klient pro prohlížení obsahu a spouštění webových aplikací. Logika se řeší na straně serveru.
Tlustý klient	Aplikace běží na zařízení a je nezávislá na serveru ve smyslu zpracování operací apod. Může běžet v „offline“ režimu, tedy bez připojení k síti.

Seznam použitých zkratek

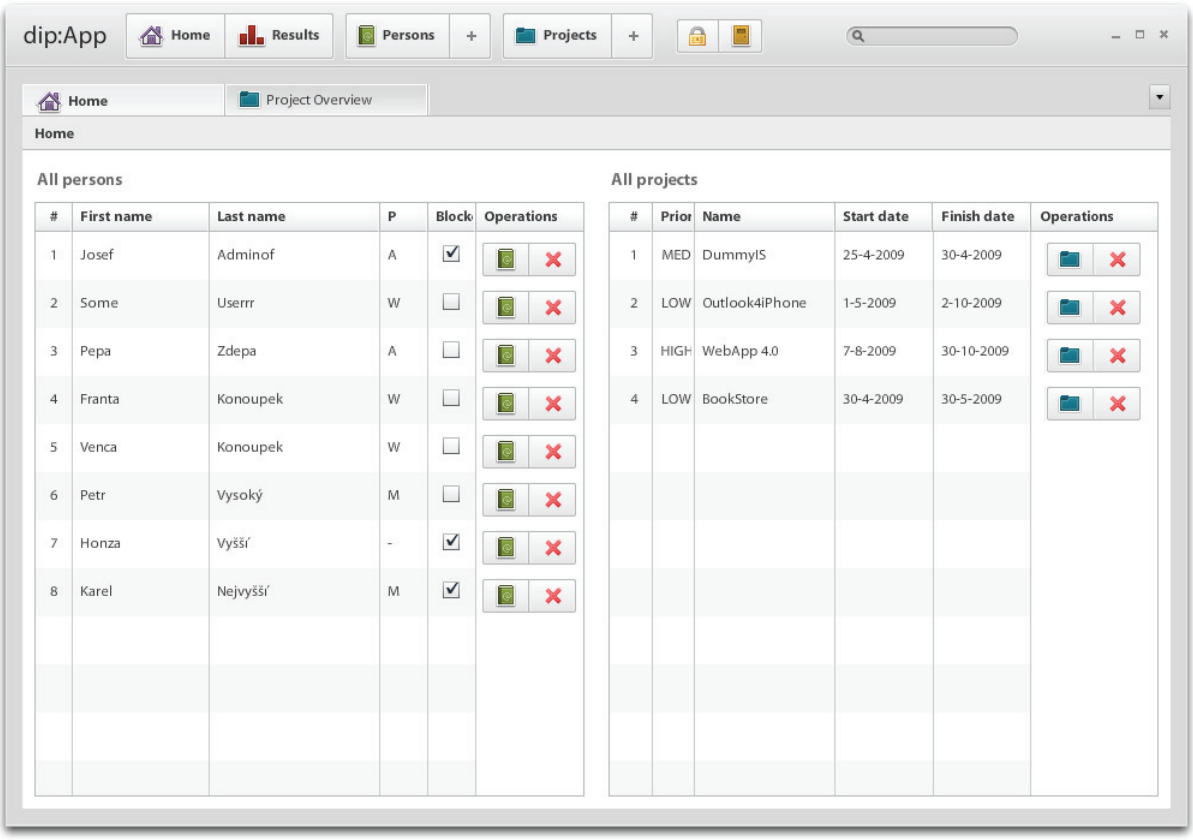
AIR	Adobe Integrated Runtime
CLR	Common Language Runtime
CMMI	Capability Maturity Model Integration
CPM	Cycles per minute
CRUD	Create, read, update, delete
DB	Databáze, database
DI	*Dependency Injection*
FDD	Feature Driven Development
GPL	General Public License
IDE	Integrated development environment
IoC	Inversion of Control
ISO	International Organization for Standardization
MDA	Model-driven architecture
MsSQL	Microsoft SQL Server
MVC	Model-View-Controller
MXML	Macromedia XML, Magic eXtensible Markup Language
PDPC	Process decision program chart
PERT	Program Evaluation and Review Technique
PgSQL	PostgreSQL
PHP	Hypertext Preprocessor
QMS	Quality Management System
RIA	Rich Internet Application
RUP	(IBM) Rational Unified Process
SDK	Software Development Kit
SWF	Shockwave Flash, Small Web Format

TDD	Test Driven Development
UI	User Interface (Uživatelské rozhraní)
UML	Unified Modeling Language
WPF/E	Windows Presentation Foundation / Everywhere
WYSIWYG	WhatYouSeeIsWhatYouGet
XAML	Extensible Application Markup Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XP	Extreme Programming

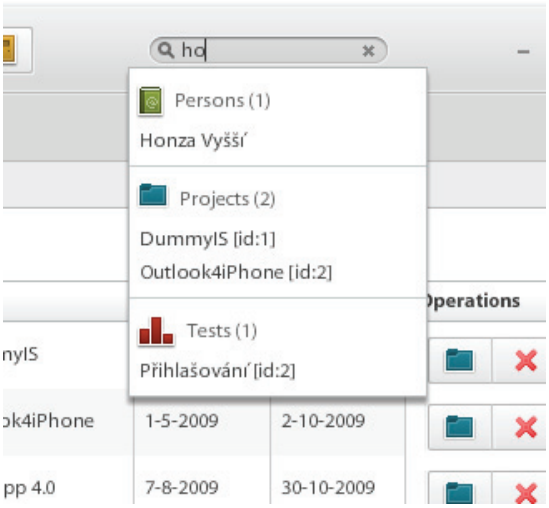
Snímky aplikace



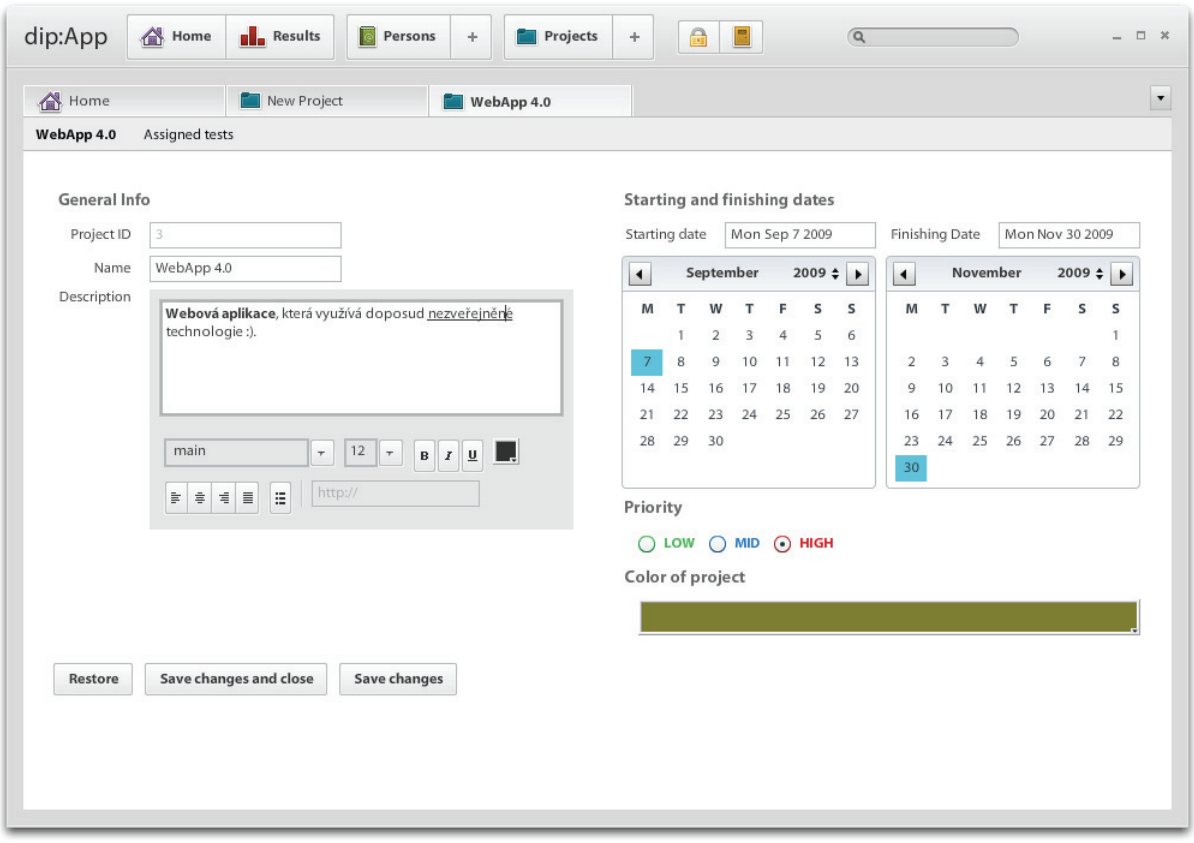
Obrázek 9-1: Přihlašovací okno se zvýrazněním nevalidních vstupů



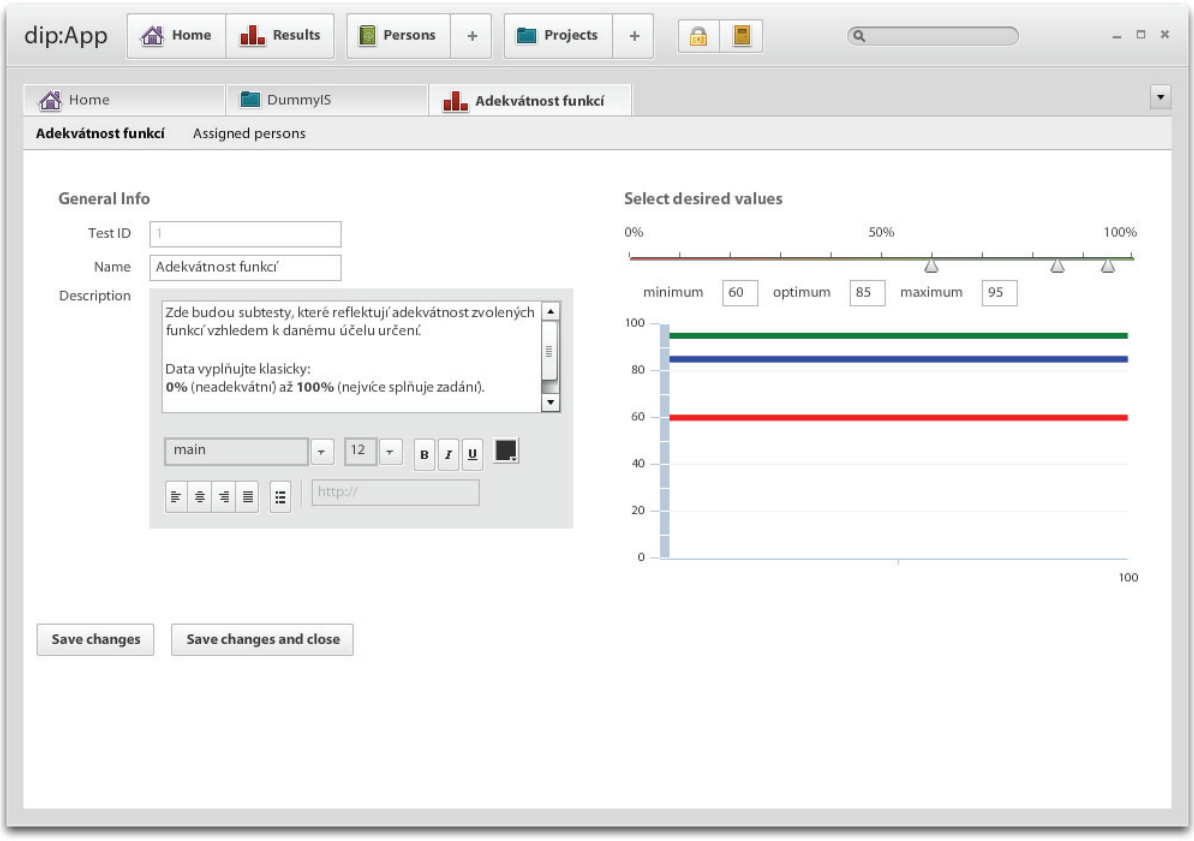
Obrázek 9-2: „Home Panel“ po přihlášení Administrátora



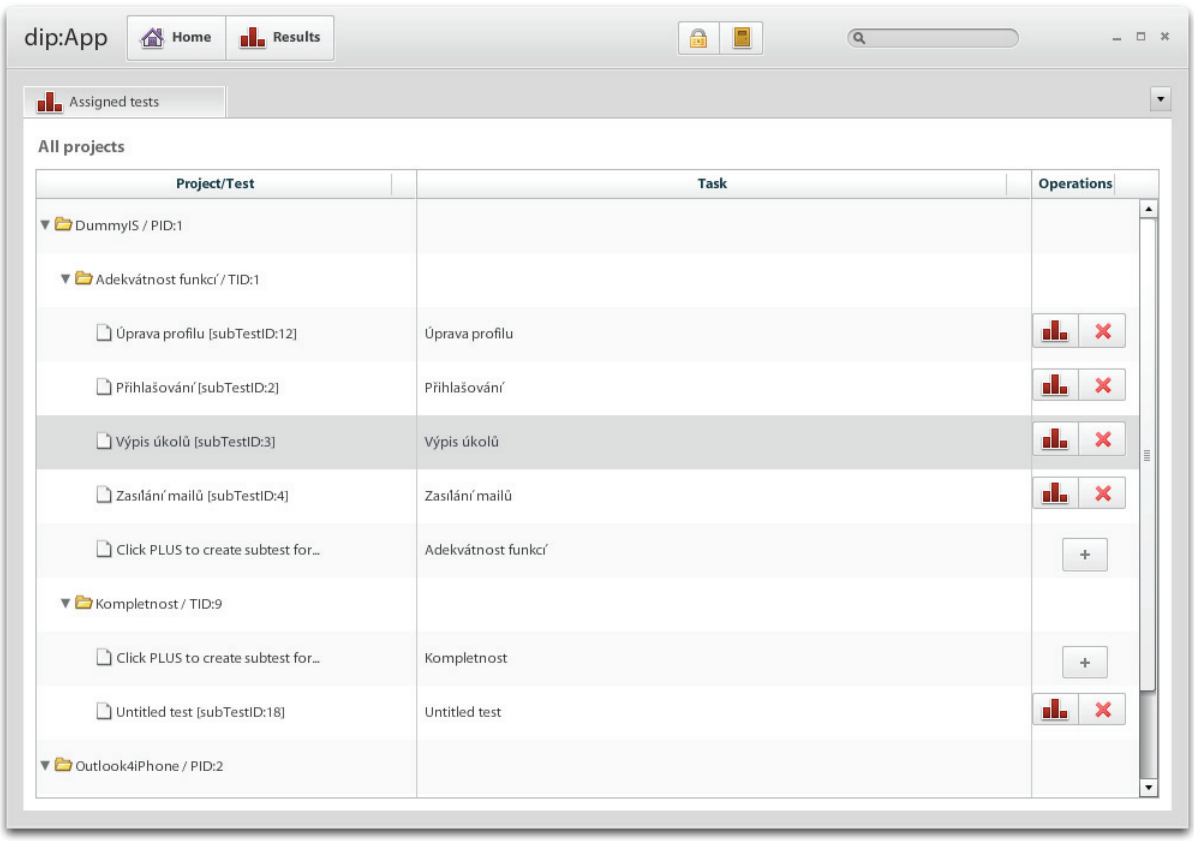
Obrázek 9-3: Panel zobrazující aktuální výsledky zadané fráze při vyhledávání (vyhledává i v popisech)



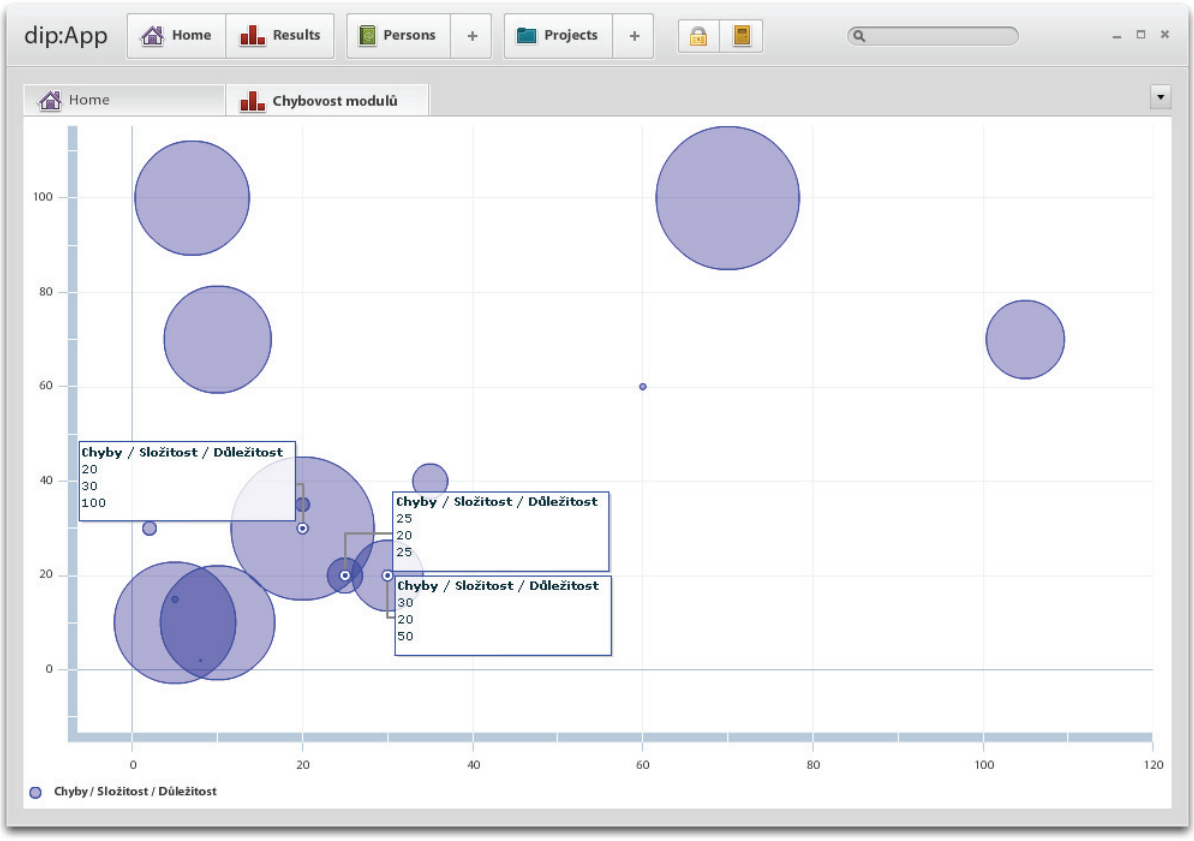
Obrázek 9-4: Úprava detailu projektu



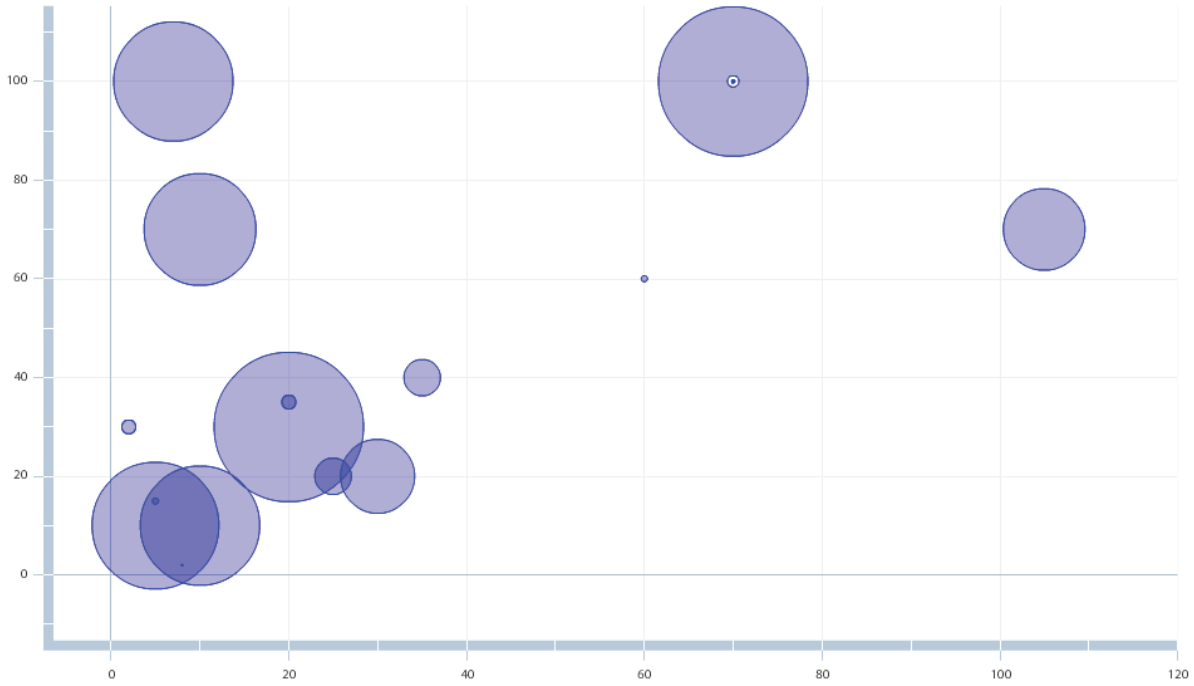
Obrázek 9-5: Úprava detailu základního testu



Obrázek 9-6: Výpis přiřazených testů

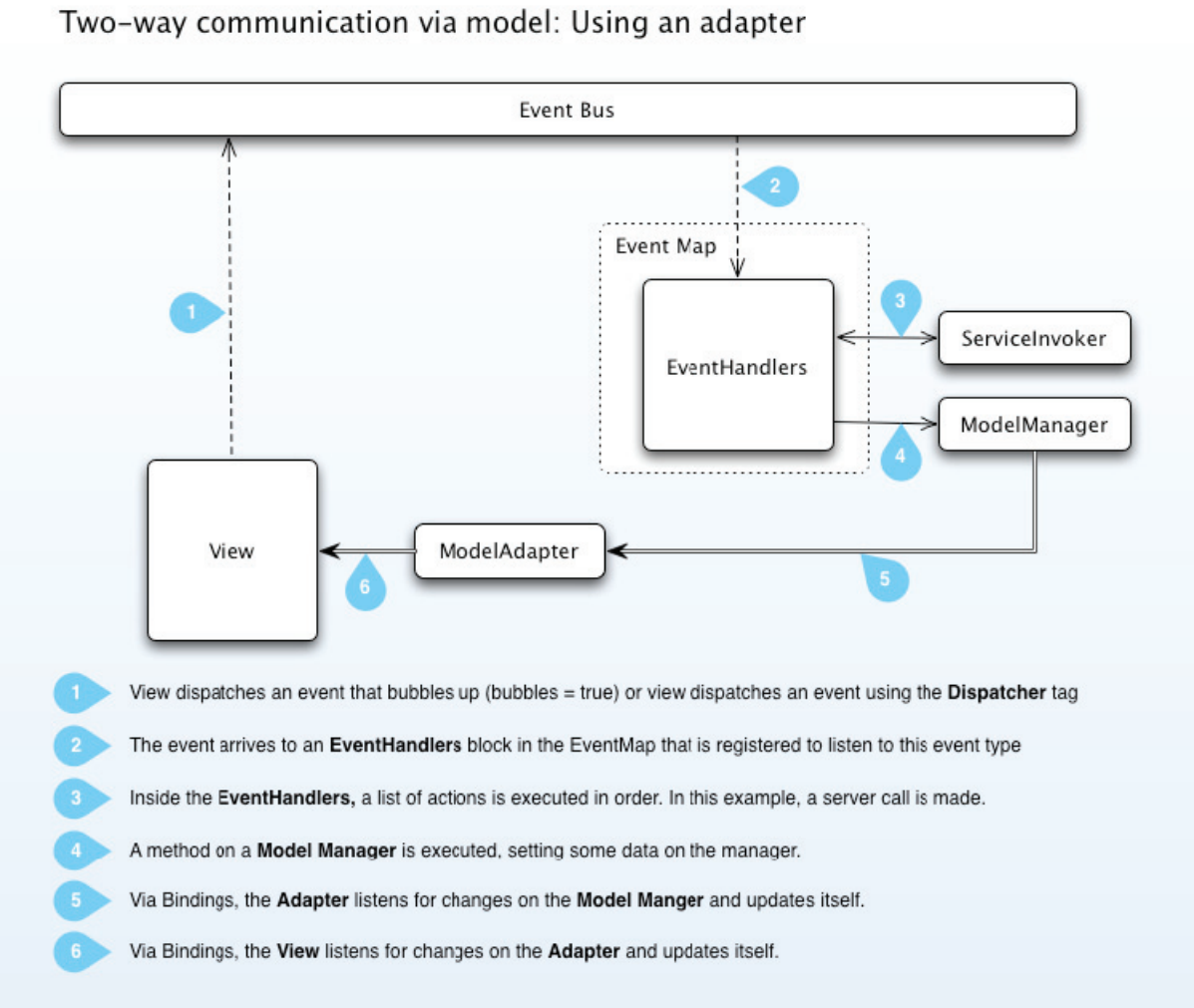


Obrázek 9-7: Zobrazení výsledků subtestů pomocí bublinového diagramu



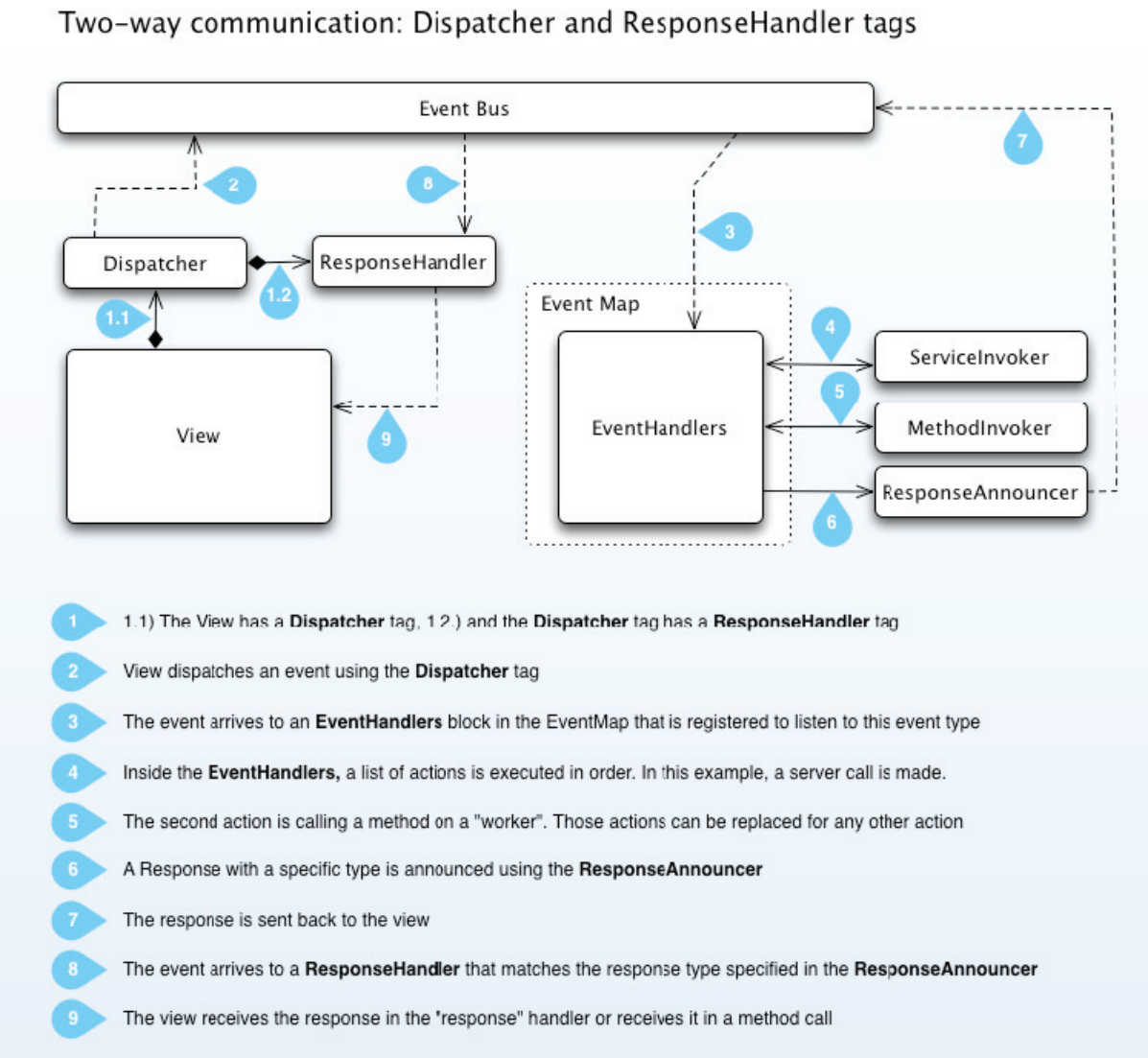
Obrázek 9-8: Stejný graf jako na Obrázek 9-7: Zobrazení výsledků subtestů pomocí bublinového diagramu přetažen přímo do aplikace Word interaktivní metodou „drag and drop“

Diagramy Frameworku Mate



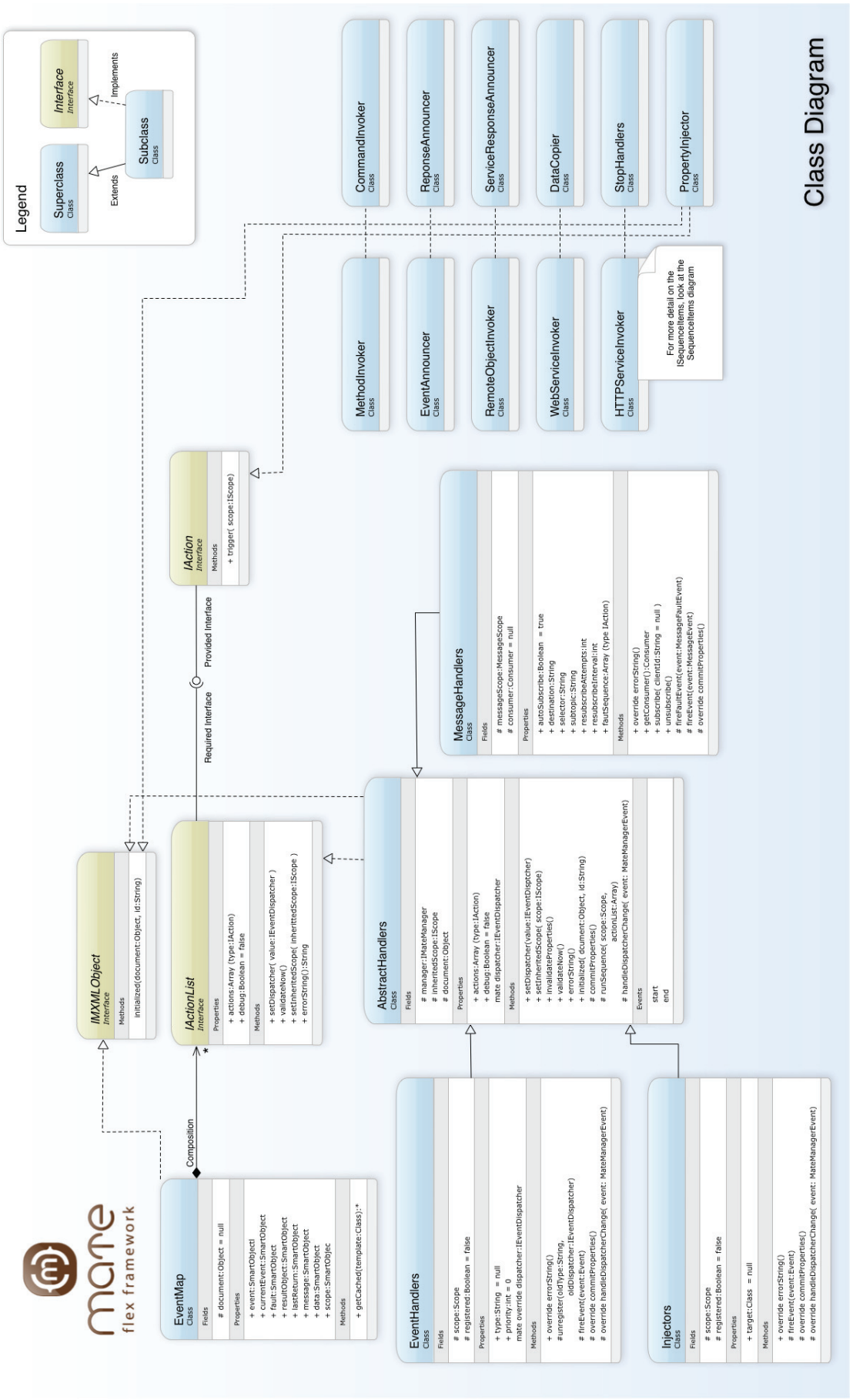
Obrázek 9-9: Dvoucestná komunikace s využitím „ModelAdapter“ [25]

[25] http://mate.asfusion.com/assets/content/diagrams/model_adapter.pdf (diagram je i na CD ve složce „doc/mate_diagrams“)

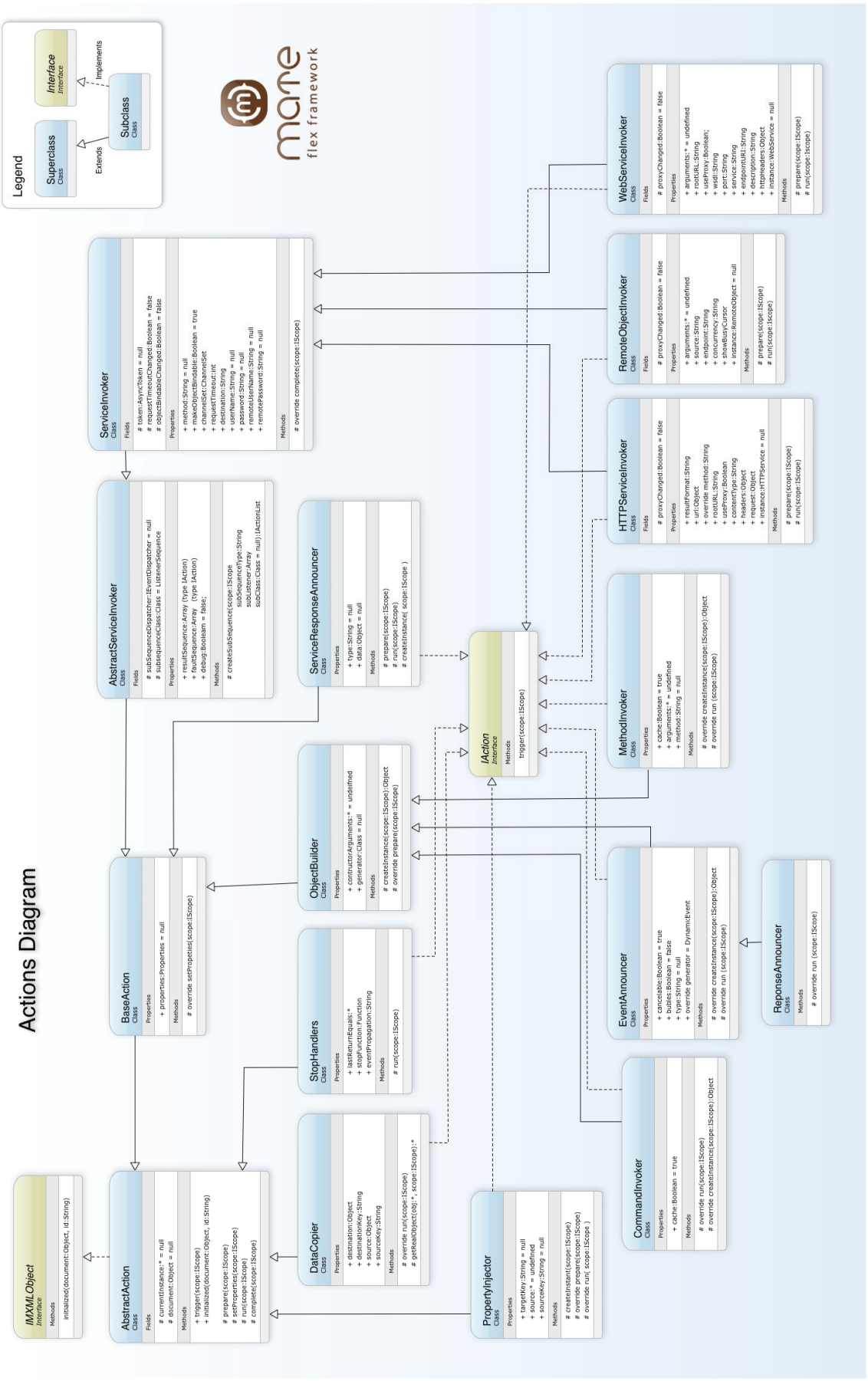


Obrázek 9-10: Dvoucestná komunikace s využitím „Dispatcher“ (odesílatele) a „ResponseHandler“ (obsluhy odpovědi) [26]

[26] http://mate.asfusion.com/assets/content/diagrams/two_ways.pdf (diagram je i na CD ve složce „doc/mate_diagrams“)



Obrázek 9-11: Diagram tříd frameworku Mate



Obrázek 9-12: Diagram „akcí“ (Actions Diagram) frameworku Mate